Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ФУНДАМЕНТАЛЬНЫЕ НАУКИ**

КАФЕДРА **ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА И МАТЕМАТИЧЕСКАЯ ФИЗИКА (ФН11)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **02.03.01 МАТЕМАТИКА И КОМПЬЮТЕРНЫЕ НАУКИ**

О Т Ч Е Т

по домашнему заданию № 2
Вариант 18

Название: ***Часть 2.1.*** **Поисковые программы с использованием файловой базы данных.**
Часть 2.2 **Древовидная рекурсия.**
Часть 2.3. **Использование бинарных деревьев для вычисления выражений.**

Дисциплина: Информатика

Студент	ФН11-22Б	02.06.21	М.Х.Хаписов
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель			Т.Н.Ничушкина
		(Подпись, дата)	(И.О. Фамилия)
		2-6-2021	

Часть 2.1. Поисковые программы с использованием файловой базы данных

Цель: закрепление материала по работе с файловыми структурами данных и применение этих структур в практике программирования

Задание: Сведения о домах на участках представлены для каждого их них этажностью, площадью, годом застройки, материалом (кирпичный, деревянный,…), стоимостью, фамилией владельца. Программа должна формировать бинарный файл типа «структура», добавлять и удалять данные, а также воспринимать каждый из перечисленных запросов и давать на него ответ.

1. Определить фамилии владельцев, имеющих дома указанной этажности.

2. Вывести данные о домах, изготовленных из указанного материала.

3. Определить фамилии владельцев, дома которых построены из дерева после указанного срока.

Для обработки файла и выбора типа обработки использовать консольное меню.

Структурная схема

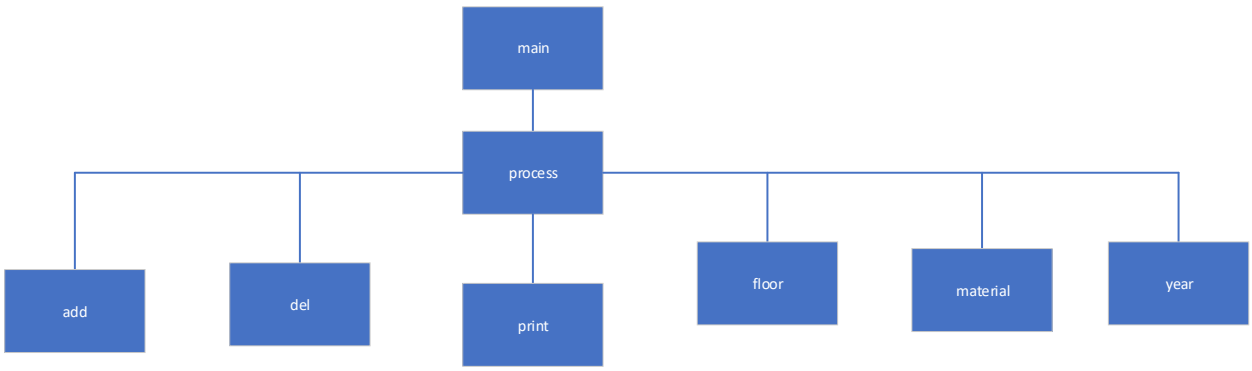
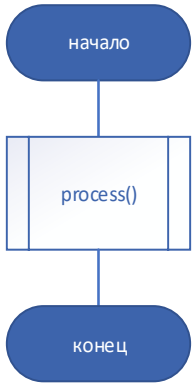
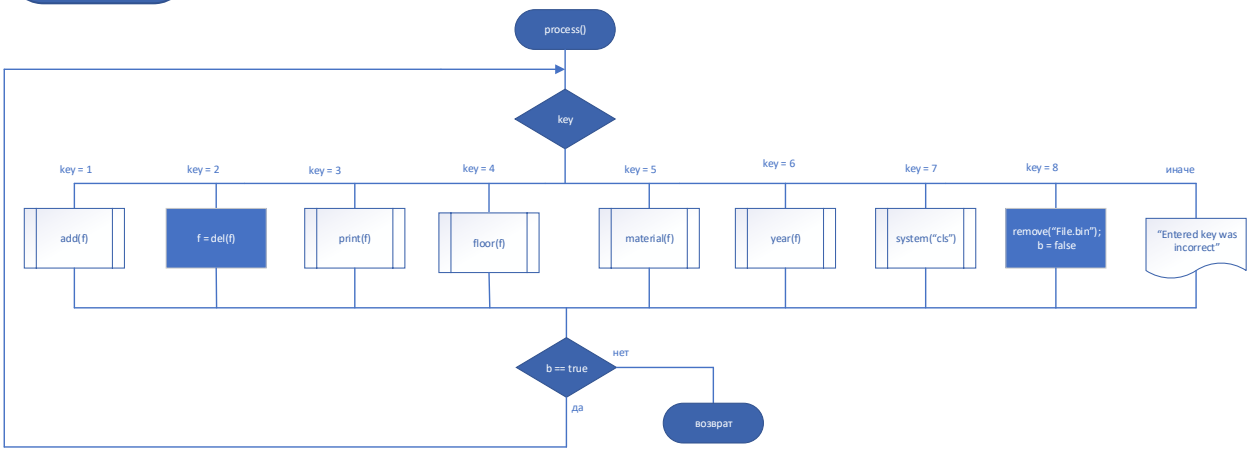
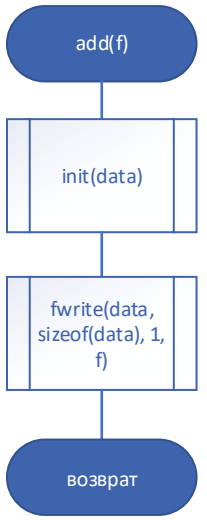
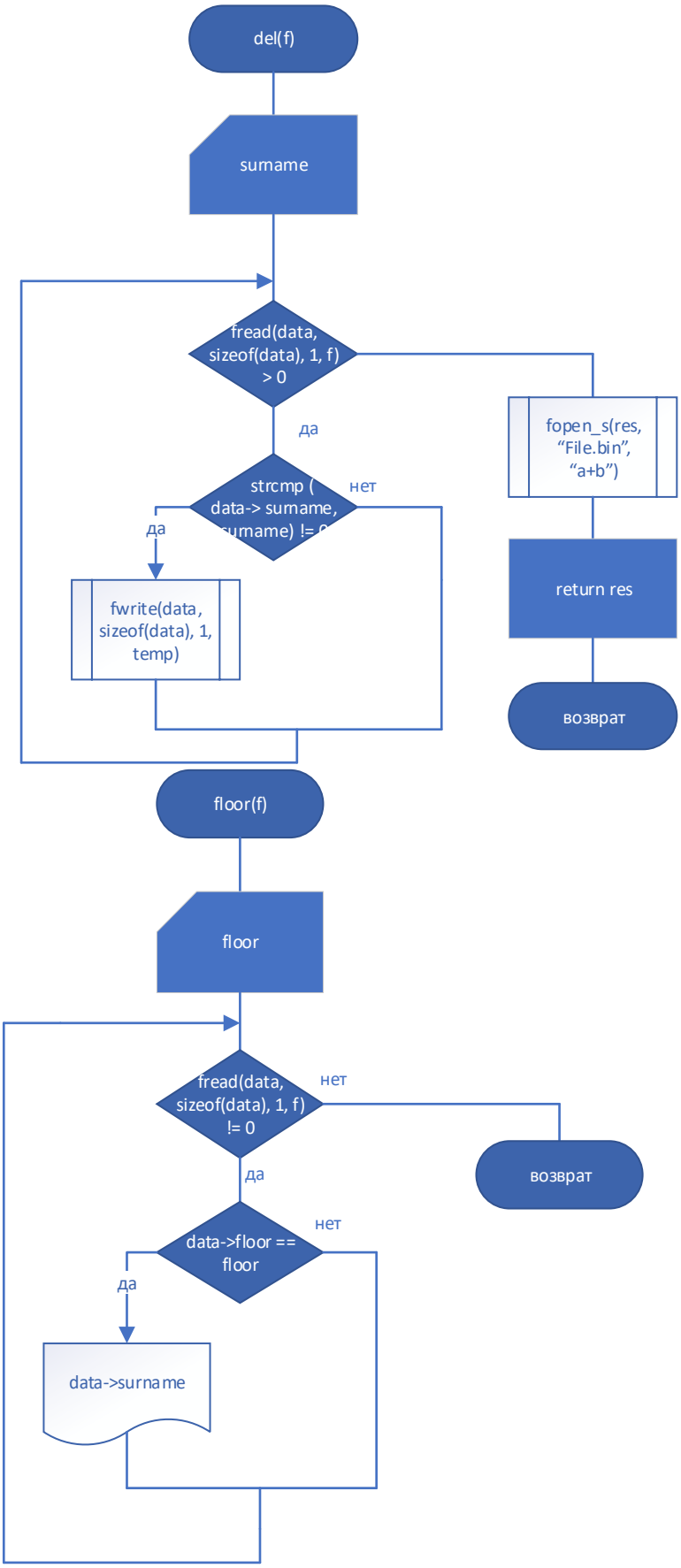


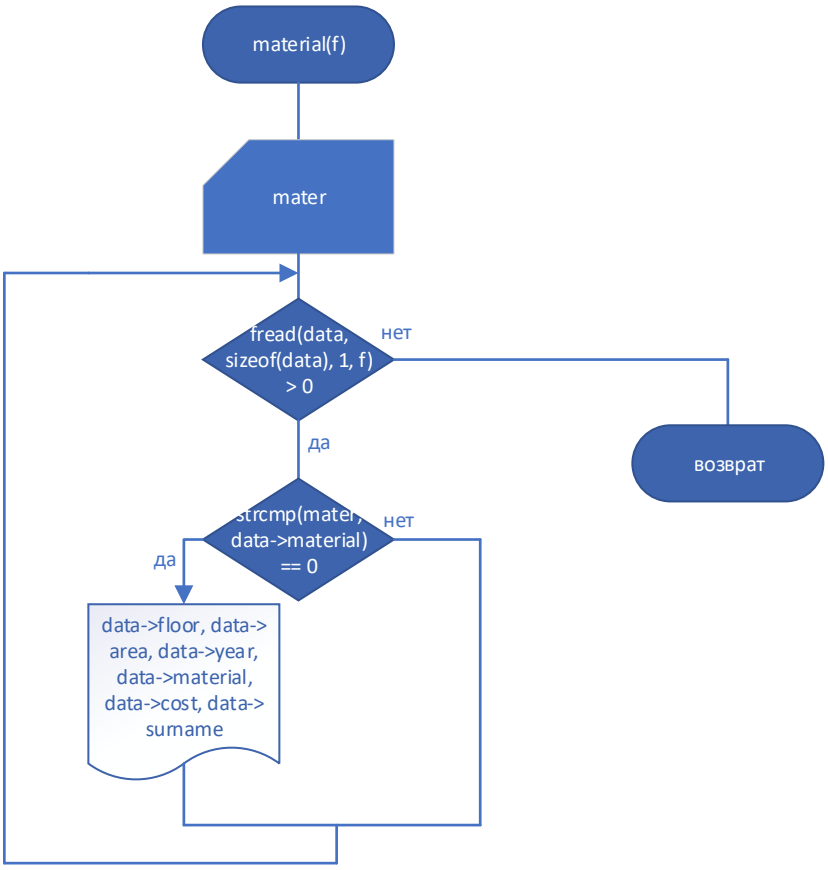
Схема алгоритма

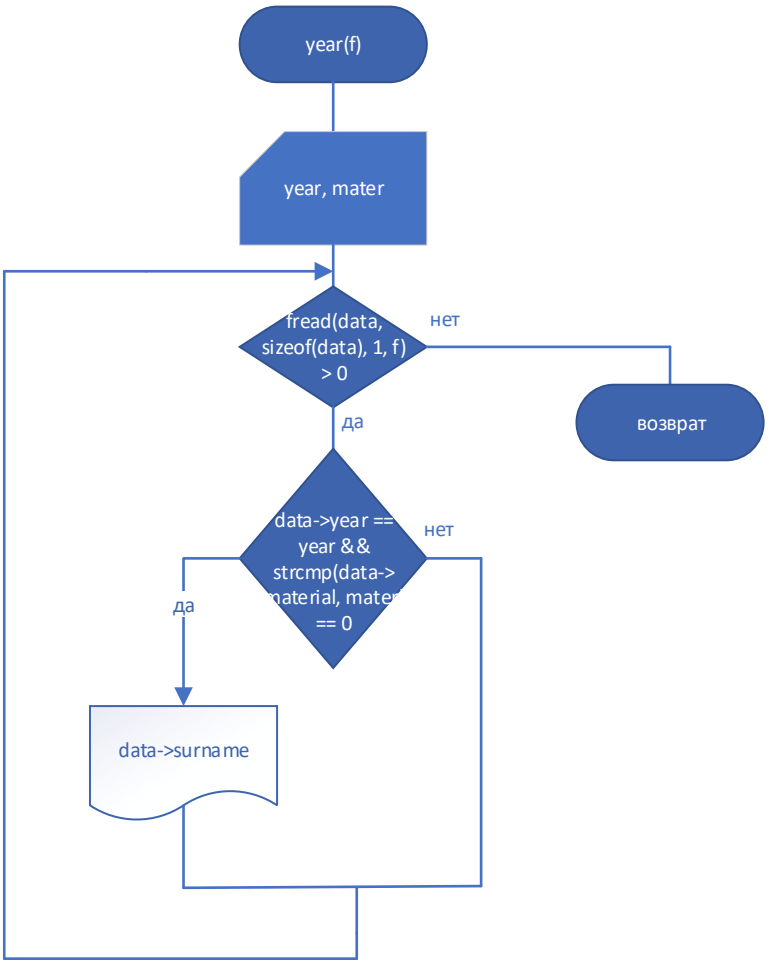







del(f)
surname
fread(data, sizeof(data), 1, f) > 0
да
нет
strcmp (data-> surname, surname) != 0
да
fwrite(data, sizeof(data), 1, temp)
fopen_s(res, “File.bin”, “a+b”)
return res
возврат
floor(f)
floor
fread(data, sizeof(data), 1, f) != 0
нет
да
возврат
data->floor == floor
нет
да
data->surname


material(f)
mater
fread(data, sizeof(data), 1, f) > 0
нет
да
возврат
strcmp(mater, data->material) == 0
нет
да
data->floor, data->area, data->year, data->material, data->cost, data->surname


year(f)
year, mater
fread(data, sizeof(data), 1, f) > 0
нет
да
возврат
data->year == year && strcmp(data->material, mater == 0
нет
да
data->surname

Текст программы

```
#include <iostream>

struct House {
	int floor;
	double area;
	int year;
	char material[64];
	int cost;
	char surname[64];
};

void init(House* data) {
	std::cout << "\nInput floor\n";
	std::cin >> data->floor;

	std::cout << "Input area\n";
	std::cin >> data->area;

	std::cout << "Input year\n";
	std::cin >> data->year;

	std::cout << "Input material\n";
	std::cin >> data->material;

	std::cout << "Input cost\n";
	std::cin >> data->cost;

	std::cout << "Input surname\n";
	std::cin >> data->surname;
}

void add(FILE* f) {
	House* data = new House;
	init(data);
	rewind(f);
	fwrite(data, sizeof(*data), 1, f);
}

FILE* del(FILE* f) {
	FILE* temp;
	fopen_s(&temp, "File2.bin", "w+b");
	if (temp != NULL) {
		char* surname = new char[64];

		std::cout << "Input a surname to delete\n";
		std::cin >> surname;

		House* data = new House;
		rewind(f);
		while (fread(data, sizeof(*data), 1, f) > 0) {
			if (strcmp(data->surname, surname) != 0)
				fwrite(data, sizeof(*data), 1, temp);
		}

		delete data;
		delete[] surname;

		fclose(f);
		fclose(temp);
		remove("File.bin");
		rename("File2.bin", "File.bin");
		FILE* res;
```

```
			fopen_s(&res, "File.bin", "a+b");
			return res;
		}
		return f;
}

void print(FILE* f) {
		if (f != NULL) {
			House* data = new House;

			rewind(f);
			while (fread(data, sizeof(*data), 1, f) > 0)
				std::cout << "\nFloor: " << data->floor << "\n Area: " << data->area
<< "\n Year: " << data->year << "\n Material: " << data->material << "\n Cost: " << data-
>cost << "\n Surname: " << data->surname << std::endl;

			delete data;
		}
}

void floor(FILE* f) {
		if (f != NULL) {
			House* data = new House;
			int floor;

			std::cout << "\nInput the floor to check data about it\n";
			std::cin >> floor;

			rewind(f);
			while (fread(data, sizeof(*data), 1, f) != 0) {
				if (data->floor == floor)
					std::cout << "\nSurname: " << data->surname << std::endl <<
std::endl;
			}

			delete data;
		}
}

void material(FILE* f) {
		if (f != NULL) {
			House* data = new House;
			char* mater = new char[64];
			std::cout << "\nInput the material to check data about it\n";
			std::cin >> mater;

			rewind(f);
			while (fread(data, sizeof(*data), 1, f) > 0) {
				if (strcmp(mater, data->material) == 0)
					std::cout << "\nFloor: " << data->floor << "\n Area: " <<
data->area << "\n Year: " << data->year << "\n Material: " << data->material << "\n Cost:
" << data->cost << "\n Surname: " << data->surname << std::endl;
			}

			delete data;
			delete[] mater;
		}
}

void year(FILE* f) {
		if (f != NULL) {
			House* data = new House;
			int year;
			char* mater = new char[64];
```

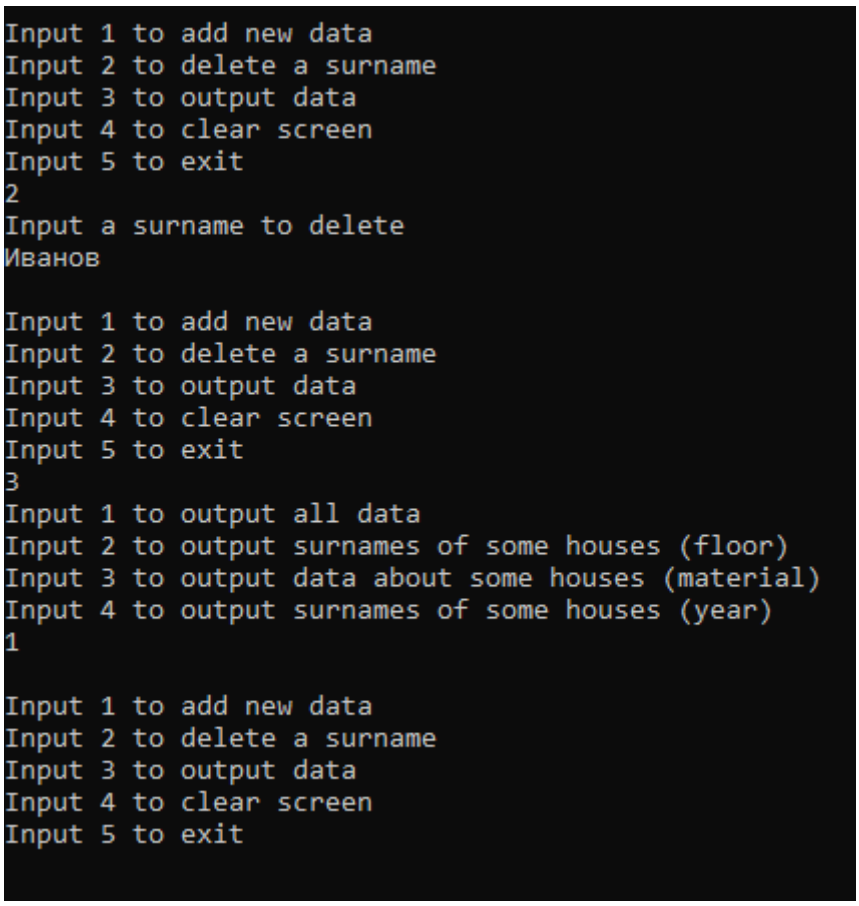
```
			std::cout << "\nInput the year and the material to check data about it\n";
			std::cin >> year >> mater;

			rewind(f);
			while (fread(data, sizeof(*data), 1, f) > 0) {
				if (data->year == year && strcmp(data->material, mater) == 0)
					std::cout << "\nSurname: " << data->surname << std::endl;
			}

			delete data;
		}
	}

	void process() {
		FILE* f;
		fopen_s(&f, "File.bin", "a+b");
		int key;
		bool b = true;
		do {
			std::cout << "\nInput 1 to add new data\n";
			std::cout << "Input 2 to delete a surname\n";
			std::cout << "Input 3 to output all data\n";
			std::cout << "Input 4 to output surnames of some houses(floor)\n";
			std::cout << "Input 5 to output data about some houses(material)\n";
			std::cout << "Input 6 to output surnames of some houses(year)\n";
			std::cout << "Input 7 to clear screen\n";
			std::cout << "Input 8 to exit\n";
			std::cin >> key;
			switch (key) {
				case 1: add(f); break;
				case 2: f = del(f); break;
				case 3: print(f); break;
				case 4: floor(f); break;
				case 5: material(f); break;
				case 6: year(f); break;
				case 7: system("cls"); break;
				case 8: remove("File.bin"); b = false; break;
				default: std::cout << "\nEntered key was incorrect!\n";
			}
		} while (b);
	}

	void main() {
		FILE* f;
		fopen_s(&f, "File.bin", "w+b");
		if (f != NULL)
			fclose(f);
		process();
		system("pause");
	}
```

Тестирование

```
Input 1 to add new data
Input 2 to delete a surname
Input 3 to output data
Input 4 to clear screen
Input 5 to exit
1

Input floor

5
Input area
30.74
Input year
1989
Input material
деревянный
Input cost
10000
Input surname
Иванов

Input 1 to add new data
Input 2 to delete a surname
Input 3 to output data
Input 4 to clear screen
Input 5 to exit
1

Input floor
9
Input area
28.82
Input year
2000
Input material
кирпичный
Input cost
12344
Input surname
Петров
```

```
Input 1 to add new data
Input 2 to delete a surname
Input 3 to output data
Input 4 to clear screen
Input 5 to exit
3
Input 1 to output all data
Input 2 to output surnames of some houses (floor)
Input 3 to output data about some houses (material)
Input 4 to output surnames of some houses (year)

1

Floor: 5
 Area: 30.74
 Year: 1989
 Material: деревянный
 Cost: 10000
 Surname: Иванов

Floor: 9
 Area: 28.82
 Year: 2000
 Material: кирпичный
 Cost: 12344
 Surname: Петров

Input 1 to add new data
Input 2 to delete a surname
Input 3 to output data
Input 4 to clear screen
Input 5 to exit
3
Input 1 to output all data
Input 2 to output surnames of some houses (floor)
Input 3 to output data about some houses (material)
Input 4 to output surnames of some houses (year)
2

Input the floor to check data about it
5

Surname: Иванов
```

```
Input 1 to add new data
Input 2 to delete a surname
Input 3 to output data
Input 4 to clear screen
Input 5 to exit
2
Input a surname to delete
Петров

Input 1 to add new data
Input 2 to delete a surname
Input 3 to output data
Input 4 to clear screen
Input 5 to exit
3
Input 1 to output all data
Input 2 to output surnames of some houses (floor)
Input 3 to output data about some houses (material)
Input 4 to output surnames of some houses (year)

1

Floor: 5
 Area: 30.74
 Year: 1989
 Material: деревянный
 Cost: 10000
 Surname: Иванов

Input 1 to add new data
Input 2 to delete a surname
Input 3 to output data
Input 4 to clear screen
Input 5 to exit
2
Input a surname to delete
Петров

Input 1 to add new data
Input 2 to delete a surname
Input 3 to output data
Input 4 to clear screen
Input 5 to exit
3
Input 1 to output all data
Input 2 to output surnames of some houses (floor)
Input 3 to output data about some houses (material)
Input 4 to output surnames of some houses (year)
1

Floor: 5
 Area: 30.74
 Year: 1989
 Material: деревянный
 Cost: 10000
 Surname: Иванов
```



Вывод: бинарные файлы часто применяются в тех ситуациях, когда нужен доступ к произвольной части файла и тип данных содержащихся в файле элементов не является примитивным. Так, при организации базы данных мы храним в бинарном файле данные типа структура.

Ответы на контрольные вопросы

1. **Что такое база данных? Сущность? Атрибут?**
База данных — организованная в соответствии с определёнными правилами и поддерживаемая в памяти компьютера совокупность данных, характеризующая актуальное состояние некоторой предметной области и используемая для удовлетворения информационных потребностей пользователя.
Сущность — это то, о чем необходимо хранить информацию. Отсюда следует, что сущности имеют некоторые свойства(properties), соответствующие тем данным о них, которые необходимо хранить в базе.
Атрибут – это информационное отображение свойств объекта. При реализации информационной модели на каком-либо носителе информации, атрибут часто называют элементом данных, полем данных или просто полем.
2. **Какие операции с содержимым базы данных доступны пользователю?**

создание базы данных, просмотр информации о базах данных, установку параметров базы данных, переименование и удаление базы данных.

3. **Какие особенности концепции баз данных следует учитывать при организации поиска содержимого базы данных?**

Таким образом, при работе с базой данных пользователь должен знать принципы их проектирования, знать термины “сущность”, “атрибут”. Такие знания дают возможность осуществлять правильные манипуляции с данными, так как любая работа с данными, будь это выборка, удаление или вставка, выполняется с применением этих знаний.

4. **Что такое бинарный файл?**

В бинарных файлах информация хранится во внутренних форматах без преобразования к символьному виду. Запись в файл и чтение из файла осуществляется через специальный буфер, причем программист должен указывать количество обрабатываемых байт. Разделители компонентов в файле отсутствуют. Если в бинарном дисковом файле все компоненты одинаковой длины, то можно вычислить значение файлового указателя для каждого компонента, и, соответственно, осуществить прямой доступ к элементам.

5. **Какие существуют функции на языке C++ для работы с бинарными файлами?**

size_t fwrite (const void * ptr, size_t size, size_t count, FILE * stream);
size_t fread (void * ptr, size_t size, size_t count, FILE * stream);
int fseek (FILE * stream, long int offset, int origin);
int fgetpos (FILE * stream, fpos_t * pos);
long int ftell (FILE * stream);
int fsetpos (FILE * stream, const fpos_t * pos);

6. **Что такое структура?**

Структура в языке C++ представляет собой производный тип данных, который представляет какую-то определенную сущность, так же как и класс. Для определения структуры применяется ключевое слово struct, а сам формат определения выглядит следующим образом:

```
struct имя_структуры
{
компоненты_структуры
};
```

7. **Каким образом структуры влияют на хранение информации в бинарном файле?**

Если хранить данные в примитивных типах, то неясно, в каком виде или каким образом будет хранится информация в бинарном файле. Поэтому структуры позволяют хранить информацию в бинарном файле структурированно.

Часть 2.2. Древовидная рекурсия

Цель: Закрепление материала по работе с древовидной рекурсией и её применением в практике программирования.

Задача: Составить программу, используя рекурсивную процедуру (функцию).

Написать программу подсчета выдаваемой сдачи минимальным количеством купюр разного достоинства. Количество купюр каждого достоинства может быть ограничено. Сумма сдачи вводится с клавиатуры. Оттестировать программу для различных сочетаний ограничения купюр. Определить, как это влияет на число вызовов рекурсивной подпрограммы. Нарисовать дерево рекурсии для не очень большой величины сдачи. Вывести всю необходимую информацию.

Структурная схема

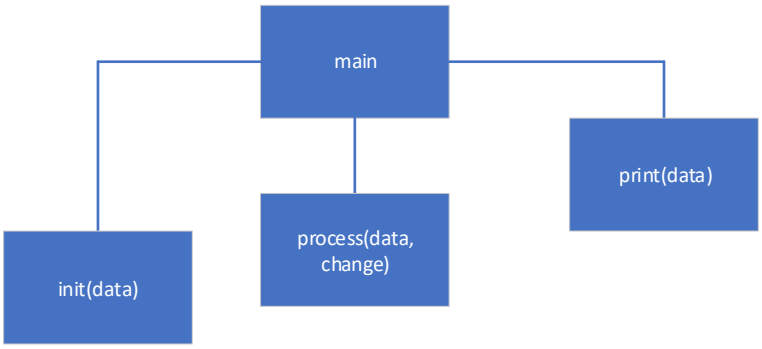
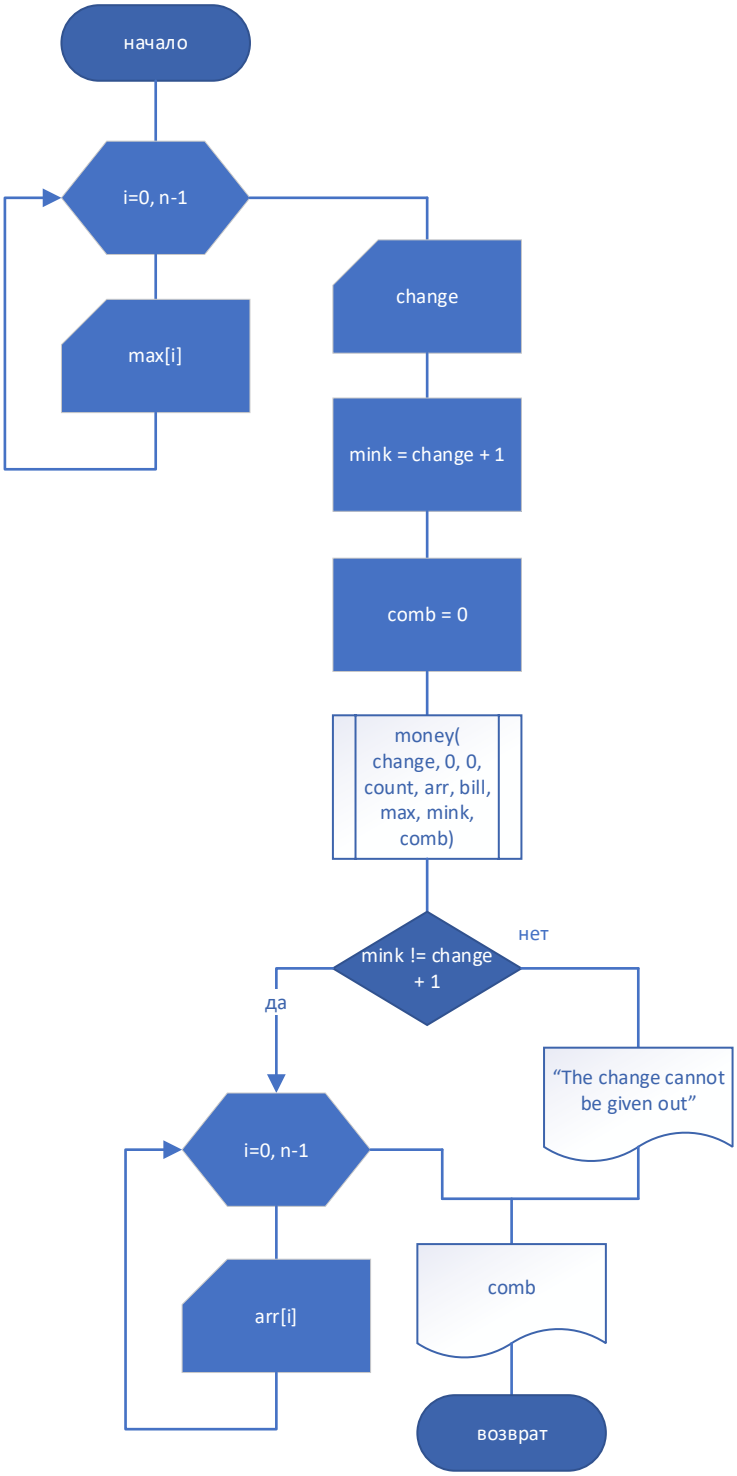
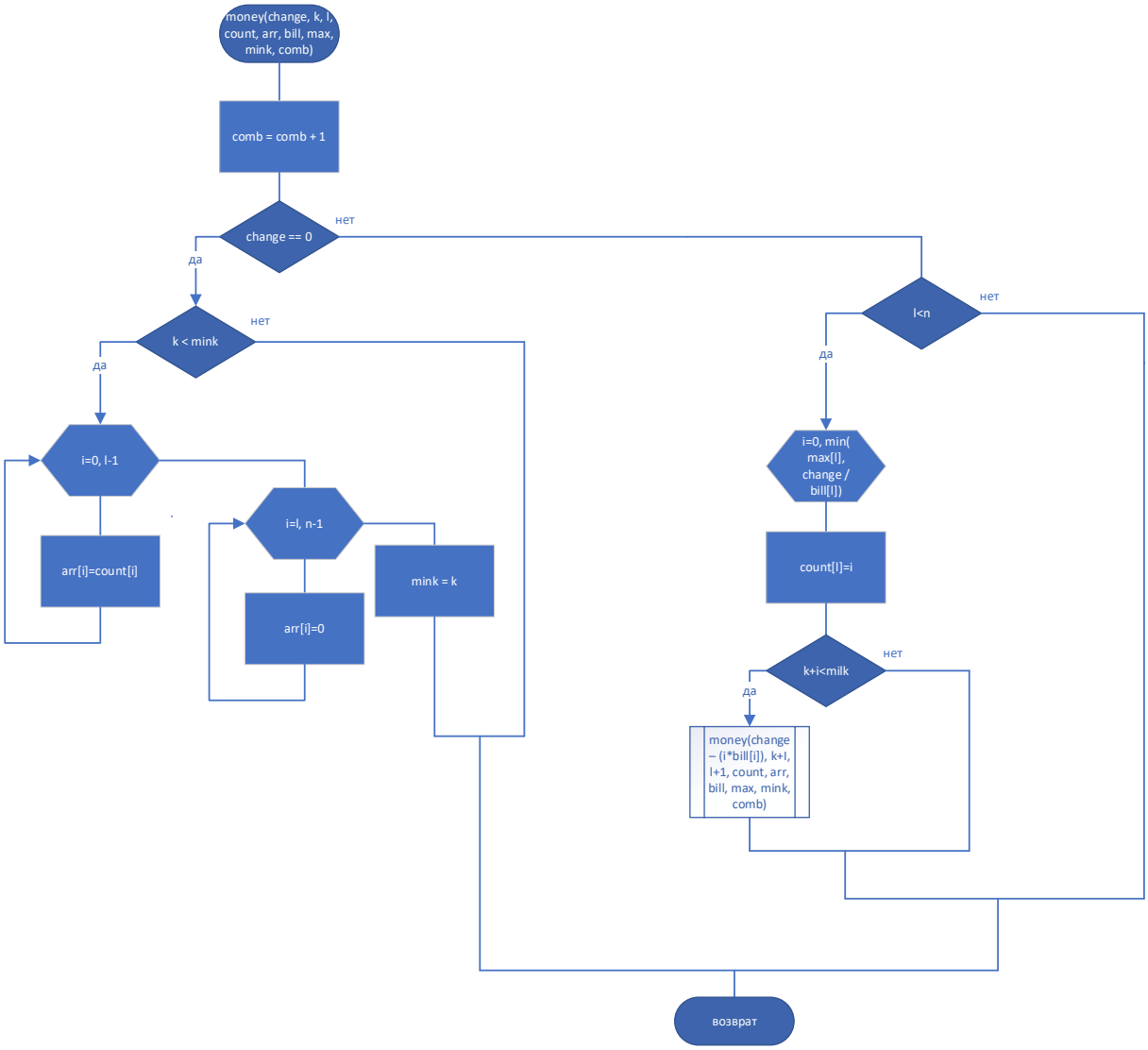


Схема алгоритма


начало
i=0, n-1
max[i]
change
mink = change + 1
comb = 0
money(change, 0, 0, count, arr, bill, max, mink, comb)
mink != change + 1
нет
да
“The change cannot be given out”
i=0, n-1
arr[i]
comb
возврат

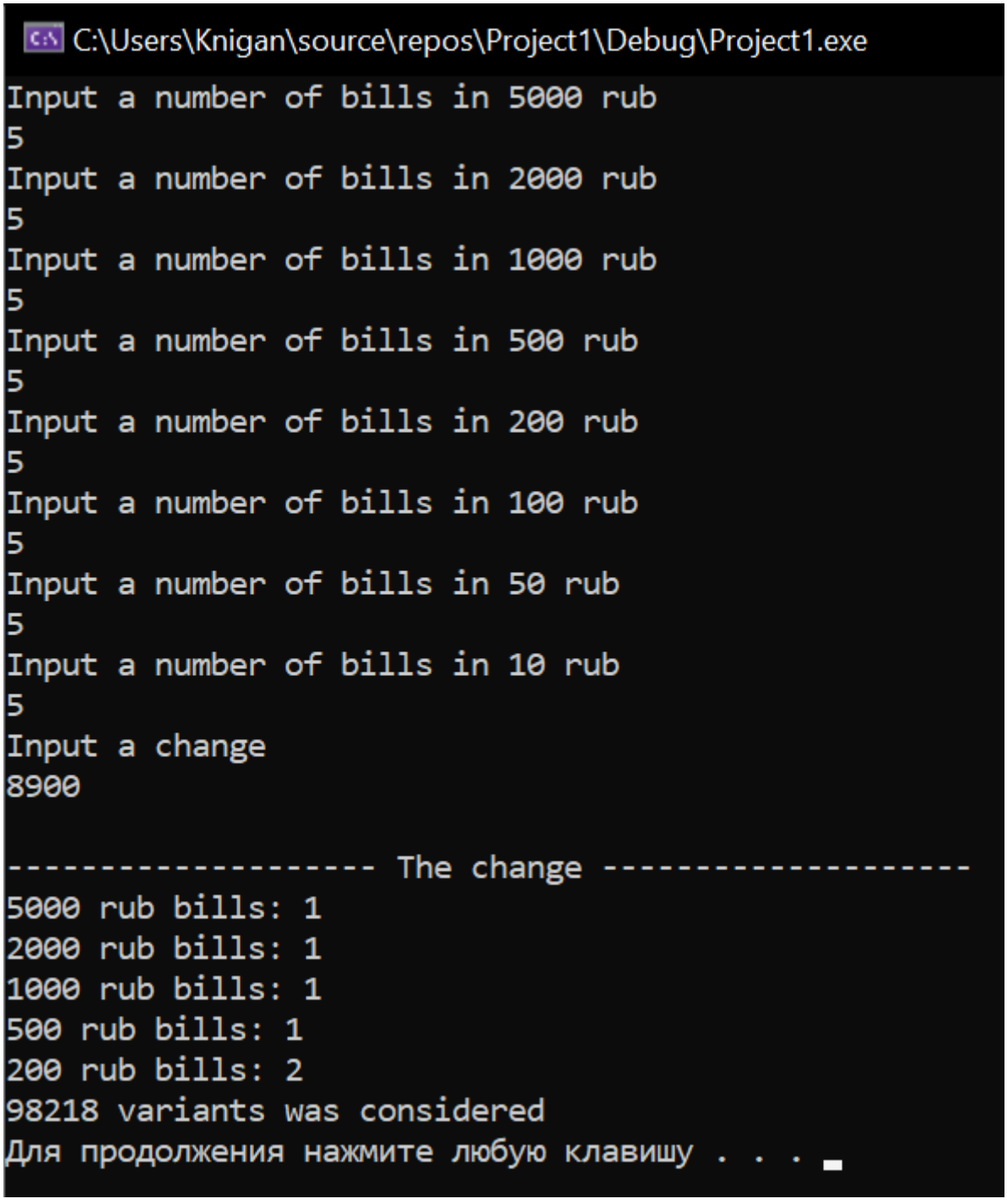


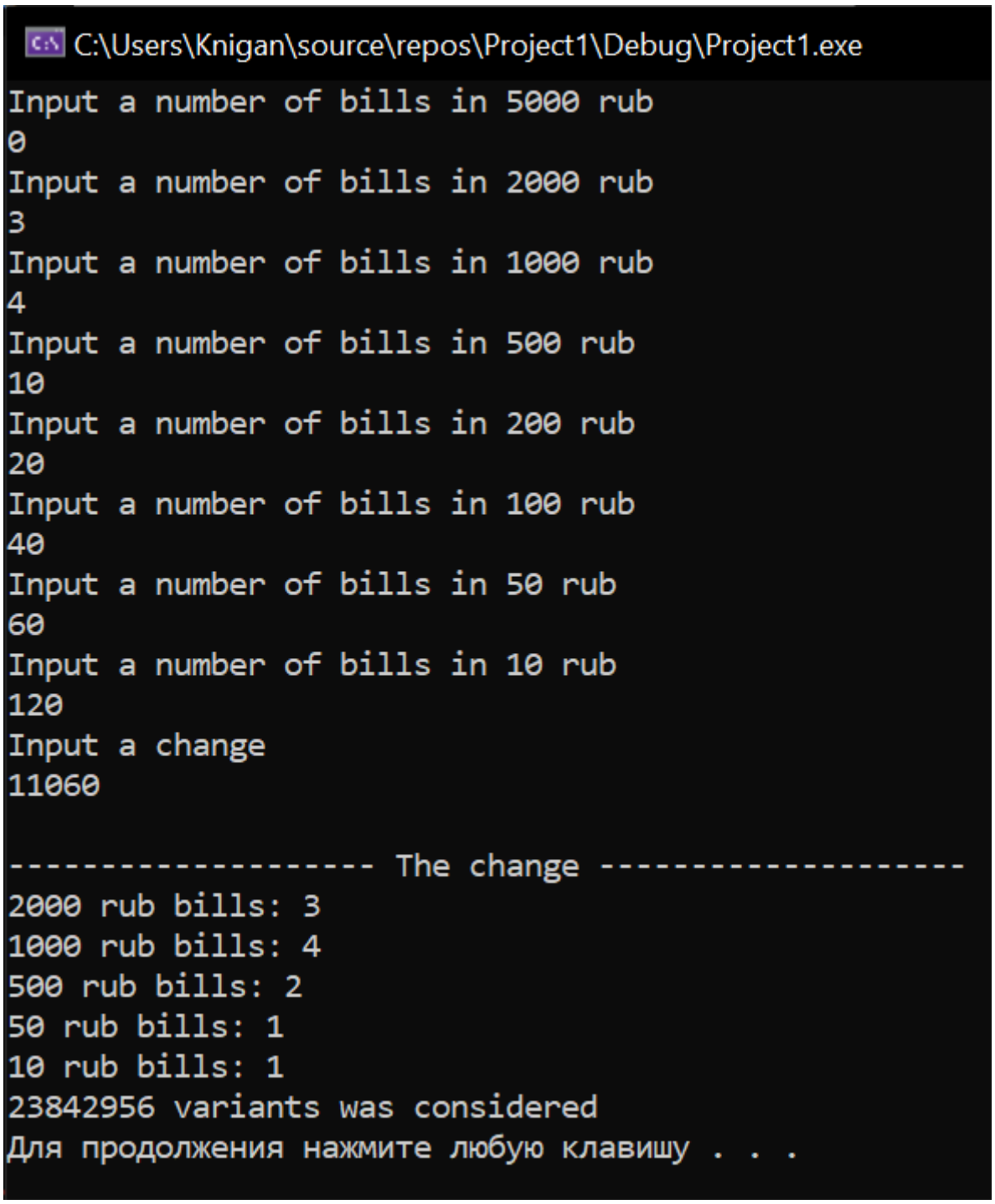
Текст программы

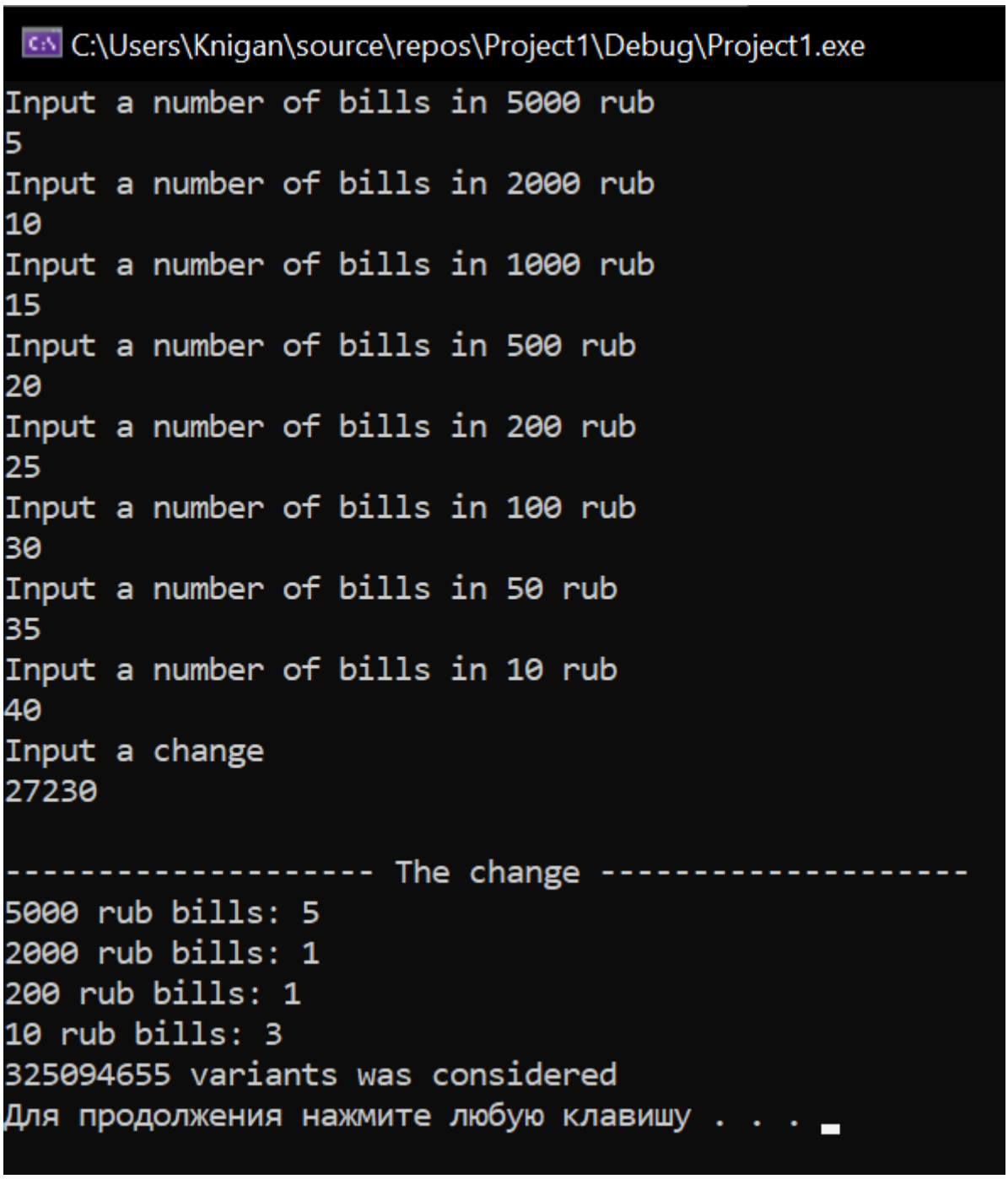
```
#define min(a, b) ((a<b) ? a : b)
#include <iostream>
const int n = 8;

void money (int change, int k, int l, int* count, int* arr, int* bill, int* max, int&
mink, int& comb) {
	++comb;
	if (change == 0) {
		if (k < mink) {
			for (int i = 0; i < l; ++i)
				arr[i] = count[i];
			for (int i = l; i < n; ++i)
				arr[i] = 0;
			mink = k;
		}
	}
	else {
		if (l < n) {
			for (int i = 0; i <= min(max[l], change / bill[l]); ++i) {
				count[l] = i;
				if (k + i < mink)
					money(change - (i * bill[l]), k + i, l + 1, count, arr,
bill, max, mink, comb);
			}
		}
```

```
	}
}

void main() {
	int count[] = { 0, 0, 0, 0, 0, 0, 0, 0 };
	int bill[] = { 5000, 2000, 1000, 500, 200, 100, 50, 10 };
	int max[n], arr[n];
	for (int i = 0; i < n; ++i) {
		std::cout << "Input a number of bills in " << bill[i] << " rub\n";
		std::cin >> max[i];
	}
	int change;
	std::cout << "Input a change\n";
	std::cin >> change;
	int mink, en, k, kn, comb;
	mink = change + 1;
	comb = 0;
	money(change, 0, 0, count, arr, bill, max, mink, comb);
	if (mink != change + 1) {
		std::cout << "\n-------------------- The change --------------------\n";
		for (int i = 0; i < n; ++i) {
			if (arr[i] != 0)
				std::cout << bill[i] << " rub bills: " << arr[i] << std::endl;
		}
	}
	else
		std::cout << "\nThe change cannot be given out\n";

	std::cout << comb << " variants was considered\n";
	system("pause");
}
```

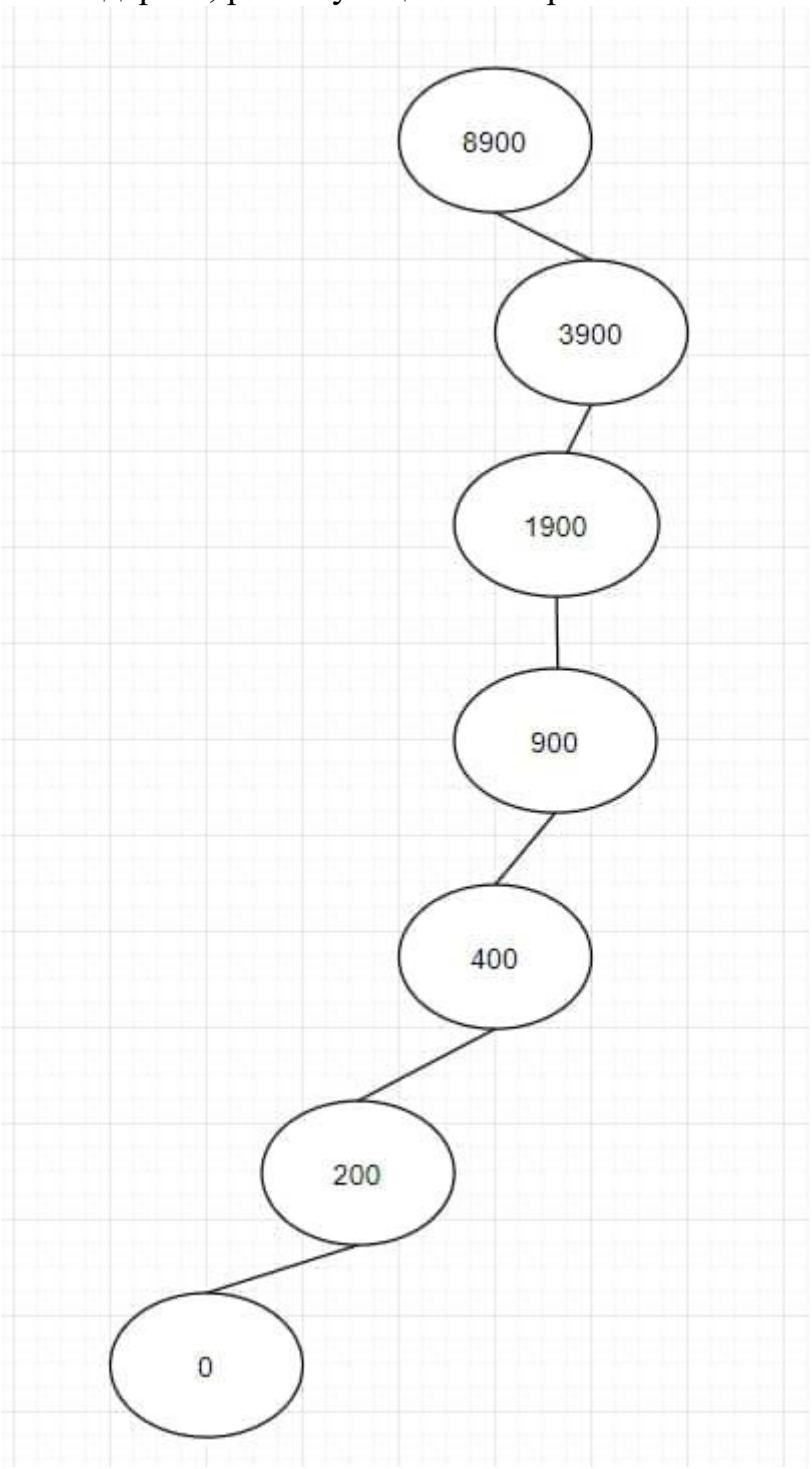
Тестирование



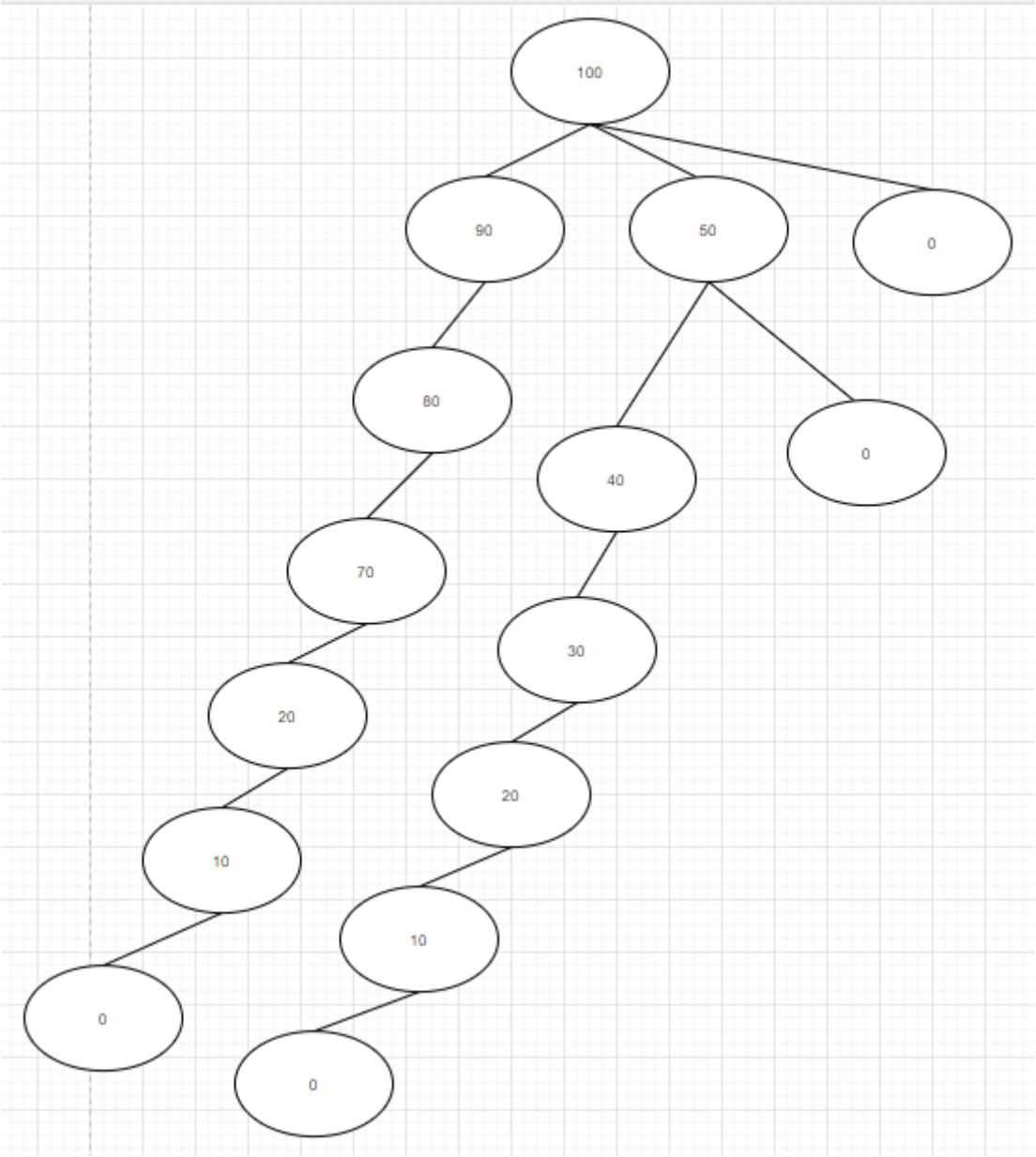

C:\Users\Knigan\source\repos\Project1\Debug\Project1.exe
Input a number of bills in 5000 rub
0
Input a number of bills in 2000 rub
3
Input a number of bills in 1000 rub
4
Input a number of bills in 500 rub
10
Input a number of bills in 200 rub
20
Input a number of bills in 100 rub
40
Input a number of bills in 50 rub
60
Input a number of bills in 10 rub
120
Input a change
11060
-------------------- The change --------------------
2000 rub bills: 3
1000 rub bills: 4
500 rub bills: 2
50 rub bills: 1
10 rub bills: 1
23842956 variants was considered
Для продолжения нажмите любую клавишу . . .


C:\Users\Knigan\source\repos\Project1\Debug\Project1.exe
Input a number of bills in 5000 rub
5
Input a number of bills in 2000 rub
10
Input a number of bills in 1000 rub
15
Input a number of bills in 500 rub
20
Input a number of bills in 200 rub
25
Input a number of bills in 100 rub
30
Input a number of bills in 50 rub
35
Input a number of bills in 10 rub
40
Input a change
27230
---------- The change ----------
5000 rub bills: 5
2000 rub bills: 1
200 rub bills: 1
10 rub bills: 3
325094655 variants was considered
Для продолжения нажмите любую клавишу . . .

Ветка дерева, реализующаяся в первом тесте



Полное дерево для числа 100



Вывод: чем меньше есть в наличии купюр высокого достоинства, тем больше купюр придётся потратить для выдачи сдачи, а следовательно, увеличивается и число вызовов рекурсивной подпрограммы.

Ответы на контрольные вопросы:

1. Дайте определение рекурсии. Из каких двух частей состоит рекурсивное утверждение?

Реку́рсия — определение, описание, изображение какого-либо объекта или процесса внутри самого этого объекта или процесса, то есть ситуация, когда объект является частью самого себя. Термин «рекурсия» используется в

различных специальных областях знаний — от лингвистики до логики, но наиболее широкое применение находит в математике и информатике.

2. Что такое взаиморекурсия? Как программно ее можно реализовать?

Прямая или явная рекурсия характеризуется существованием в теле процедуры оператора обращения к самой себе.

Косвенная или неявная рекурсия образуется в случае цепочки вызовов других процедур, которые в конечном итоге приведут к вызову начальной.

Для описания на языках программирования C и C++ прямой рекурсии никаких дополнительных операторов не требуется. При описании неявной рекурсии возникает проблема: при определении первой из нескольких взаиморекурсивных процедур или функций возникает необходимость обращения к подпрограмме, которая еще не определена, что в языке C++ недопустимо. В этом случае используется прототип функции, которая позволяет выполнять предописание:

1) сначала задаются прототипы подпрограмм с параметрами;

2) затем описываются тела этих подпрограмм (причем параметры второй раз можно уже не описывать).

3. Что такое активация рекурсивной программы и фрейм активации? Как он влияет на емкостную сложность программы? Чем опасен большой объем фрейма активации?

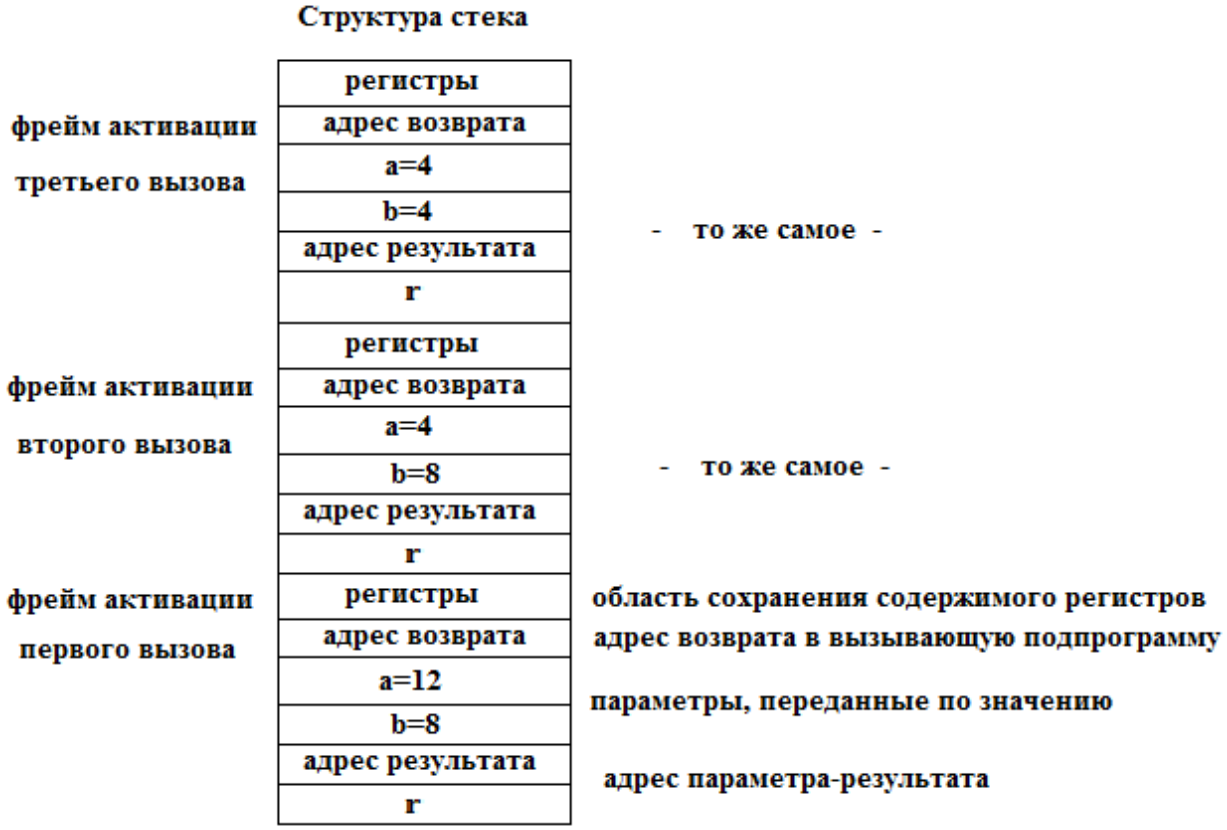
Каждое обращение к процедуре вызывает независимую активацию этой процедуры. С процедурой принято связывать некоторое множество локальных объектов (переменных, констант, типов и процедур), которые определены локально в этой процедуре, а вне ее не существуют или не имеют смысла. Совокупность данных, необходимых для одной активации называется фреймом активации.

Фрейм активации содержит независимые копии всех локальных переменных и формальных параметров процедуры, в которых оставляют "следы" операторы текущей активации.

4. Как рассчитать объем фрейма активации? Как можно уменьшить фрейм активации?

Размер фрейма активации можно примерно определить следующим образом: V= <размер области параметров>+4(адрес возврата)+2..8(для сохранения содержимого регистров)+<размер области локальных переменных>+<размер строки результата, если это функция, возвращающая результат - строку> Если вместо процедуры использовать рекурсивную функцию, то фрейм активации уменьшится, так как сократится список параметров. В нем будет отсутствовать адрес результата.

5. Дайте описание структуры рекурсивной программы.



6. Какова особенность выполнения операторов до и после рекурсивного вызова?

Когда функция вызывает сама себя, в стеке выделяется место для новых локальных переменных и параметров. Код функции работает с данными переменными. Рекурсивный вызов не создает новую копию функции. Новыми

являются только аргументы. Поскольку каждая рекурсивно вызванная функция завершает работу, то старые локальные переменные и параметры удаляются из стека и выполнение продолжается с точки, в которой было обращение внутри этой же функции. Рекурсивные функции вкладываются одна в другую как элементы подзорной трубы.

7. Что такое линейная и древовидная рекурсия? Особенности спуска и подъема каждой их этих разновидностей рекурсий.

Если подпрограмма вызывает сама себя только один раз в одной активации, то такая рекурсия называется линейной.

рекурсивный спуск, который определяется формулой рекурсивного утверждения и сопровождается вызовом рекурсивной функции (ее очередной активации). После выполнения базисного утверждения, определяющего конец рекурсивных вызовов, происходит возврат управления с одновременным вычислением значения по формулам рекурсивного утверждения. Этот процесс называется рекурсивным подъемом. Именно рекурсивный подъем формирует искомый результат.

Есть рекурсивные определения, при вычислении по которым на каждой активации необходимо вычисление двух и более выражений, требующих дополнительного вызова рекурсивной подпрограммы. Такие рекурсии получили название древовидных. количество активаций подпрограммы с древовидной рекурсией существенно возрастает, так как в памяти могут находиться одновременно активации разных веток дерева, что может привести к переполнению стека даже при небольшом фрейме активации.

8. Что такое полный и ограниченный перебор?

Существует класс задач, в которых из некоторого количества вариантов необходимо выбрать наиболее подходящий (оптимальный). Для таких задач далеко не всегда удается найти алгоритм, который позволил бы получить решение без анализа всех или большого количества комбинаций исходных данных, т.е. без осуществления перебора. Осуществление полного перебора требует много времени. Вычислительная сложность решения задач с использованием перебора обычно оценивается как 0(п) или даже 0(пп).

Для решения задач данного типа применяют две стратегии:

1) формируют сразу всю комбинацию исходных данных и выполняют ее анализ в целом;

2) генерацию и анализ комбинации осуществляют по частям.

Если имеется возможность, анализируя часть комбинации исходных данных, оценить ее перспективность с точки зрения получения решения, то вторая стратегия позволит получить результат за меньшее время за счет исключения из рассмотрения всех комбинаций, в которые входит данная часть.

Исключение бесперспективных комбинаций

9, 10. Особенности задач полного перебора. Способы уменьшения количества рассматриваемых вариантов при полном переборе.

Существует две стратегии решения задач данного типа. При первой - мы генерируем вариант, а затем определяем его пригодность. При второй -

вариант генерируется по одному элементу, и проверка пригодности осуществляется после
добавления каждого элемента. Поскольку вычислительная сложность задач данного типа - nn, то есть при увеличении размерности задачи время вычислений возрастает как nn, вторая стратегия предпочтительней.

11. В чем преимущество использования рекурсии при решении задач полного и ограниченного перебора.

В этом случае программный счетчик не поможет, так как он генерирует комбинации с повтором значений, и их нужно исключать из рассмотрения. Решение такой задачи упрощается при использовании древовидной рекурсии.

12. Дайте определение бинарного дерева.

иерархическая структура данных, в которой каждый узел имеет не более двух потомков (детей). Как правило, первый называется родительским узлом, а дети называются левым и правым наследниками.

13. Чем отличается рекурсивная процедура построения бинарного дерева?

В математике бинарным (двоичным) деревом называют конечное множество вершин, которое либо пусто, либо состоит из корня и не более чем двух непересекающихся бинарных деревьев, называемых левым и правым поддеревьями данного корня.

Таким образом, каждая вершина бинарного дерева может включать одно или два поддерева или не включать поддеревьев вовсе. Первое поддерево обычно называют левым, а второе – правым.

14. В чем особенности процедуры удаления вершины бинарного дерева?

Непосредственное удаление вершины реализуется в зависимости от того, какая вершина удаляется:

а) Удаляемая вершина не содержит поддеревьев (лист): просто удаляем ссылку на вершину из корня соответствующего поддерева

Удаляемая вершина содержит одну ветвь: для удаления необходимо скорректировать соответствующую ссылку в корне, заменив адрес удаляемой вершины адресом вершины, из нее выходящей.

Удаляемая вершина содержит две ветви: в этом случае нужно найти подходящую вершину, которую можно вставить на место удаляемой, причем эта подходящая вершина должна легко перемещаться. Такая вершина всегда существует: это либо *самый правый* элемент *левого* поддерева, либо *самый левый* элемент *правого* поддерева удаляемой вершины

Часть 2.3. Использование бинарных деревьев для вычисления выражений.

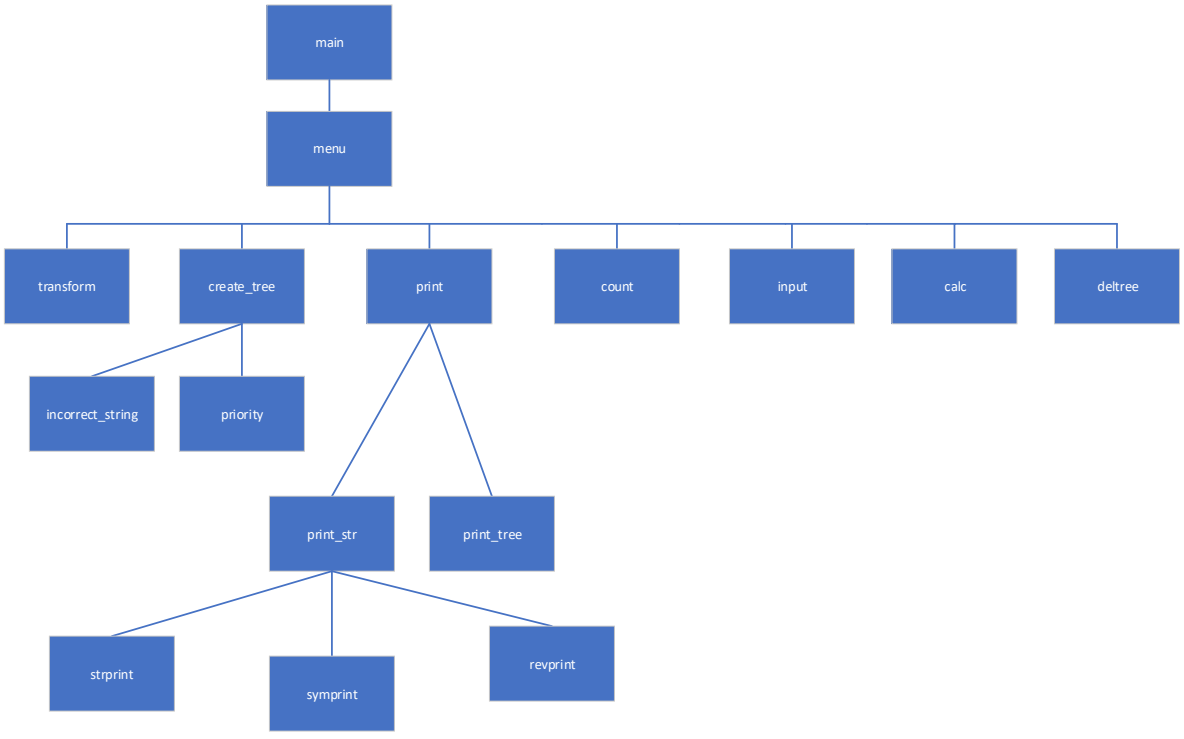
Цель:

закрепление материала по динамическим структурам данным и применение этих структур в практике программирования.

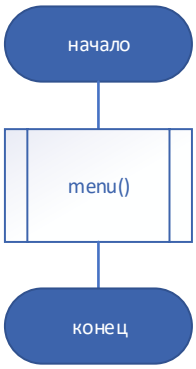
Задание:

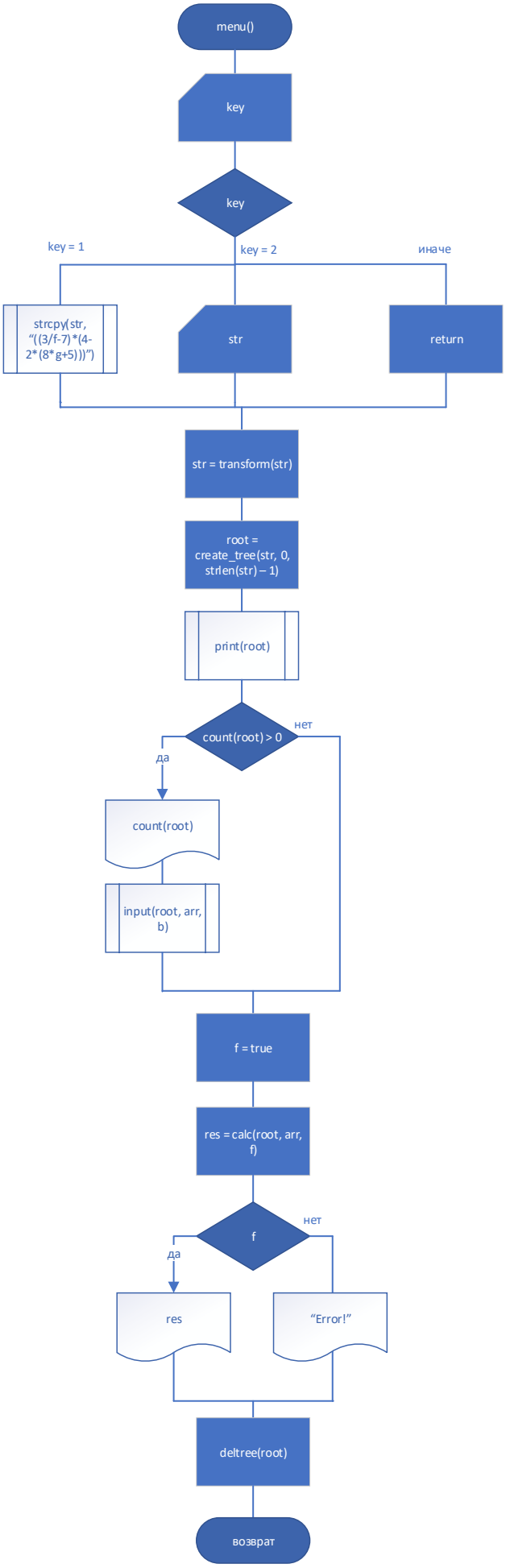
Для заданного арифметического выражения построить дерево арифметического разбора, написать алгоритм подсчета арифметического выражения, организовав ввод переменных с клавиатуры и с помощью датчика случайных чисел в соответствии с заданным вариантом. Вывести дерево и результат расчета на экран. Провести тестирование программы. Выражение: (3/f-7)*(4-2*(8*g+5))

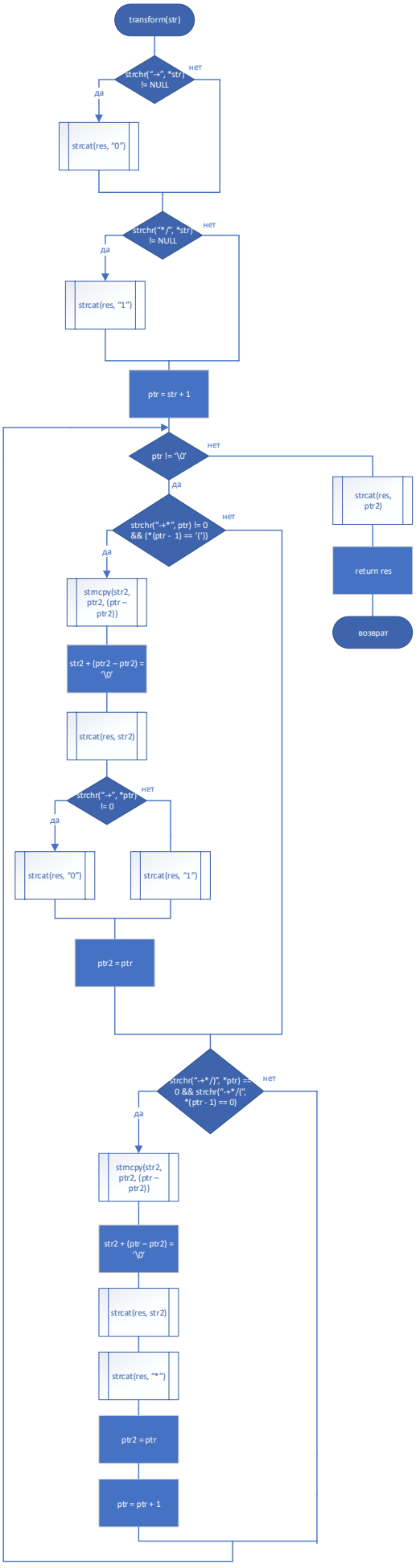
Структурная схема

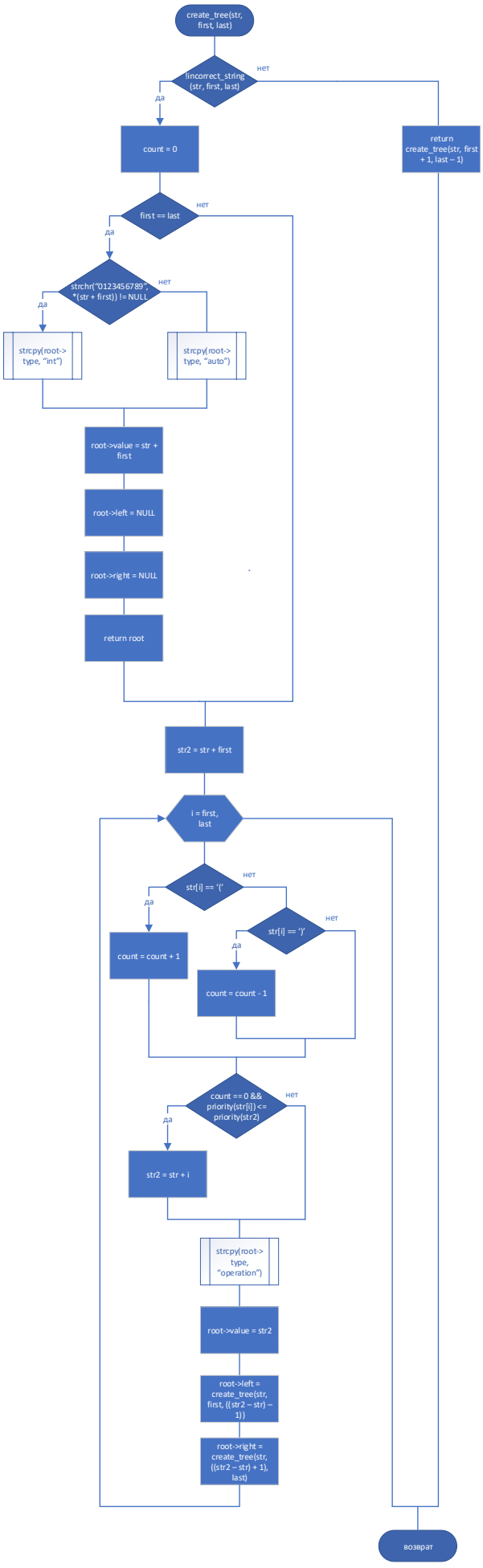


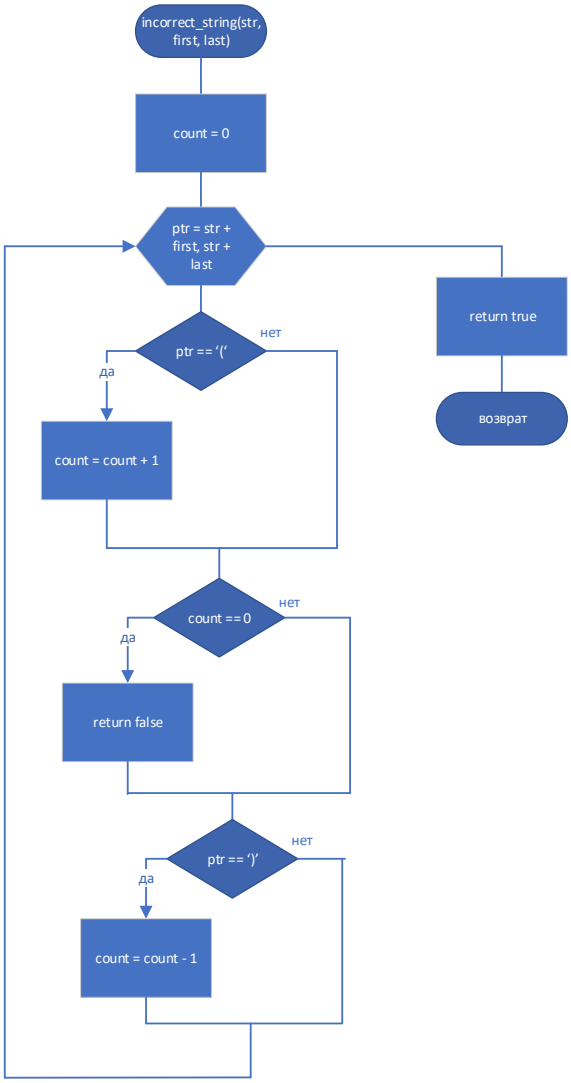
Схемы алгоритмов

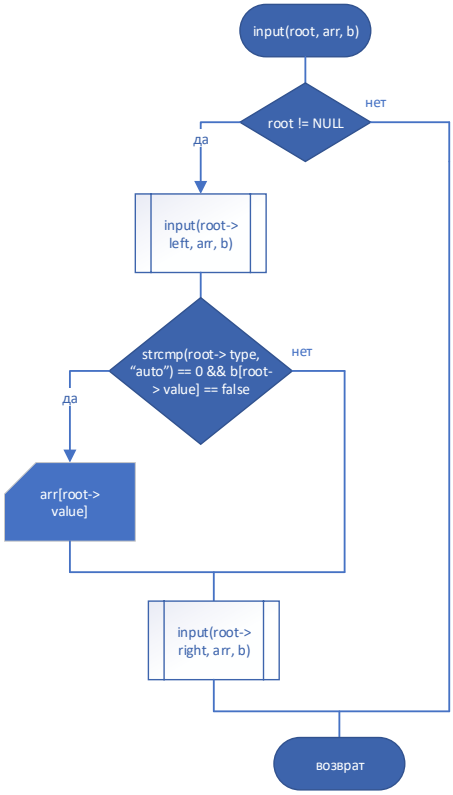

начало
menu()
конец

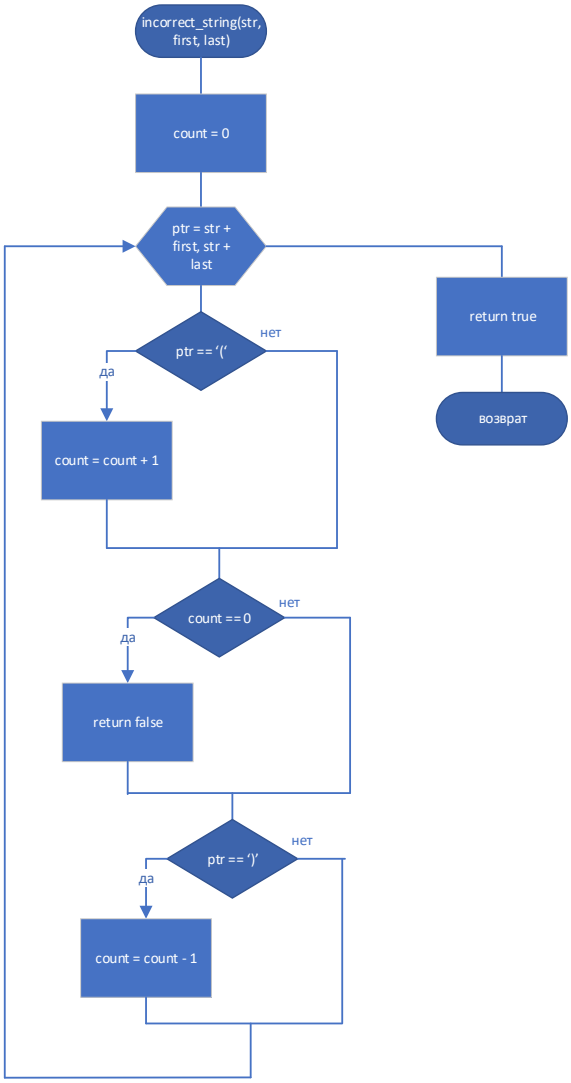

menu()
key
key
key = 1
key = 2
иначе
strcpy(str, “((3/f-7)*(4-2*(8*g+5)))”)
str
return
str = transform(str)
root = create_tree(str, 0, strlen(str) – 1)
print(root)
count(root) > 0
нет
да
count(root)
input(root, arr, b)
f = true
res = calc(root, arr, f)
f
нет
да
res
“Error!”
deltree(root)
возврат

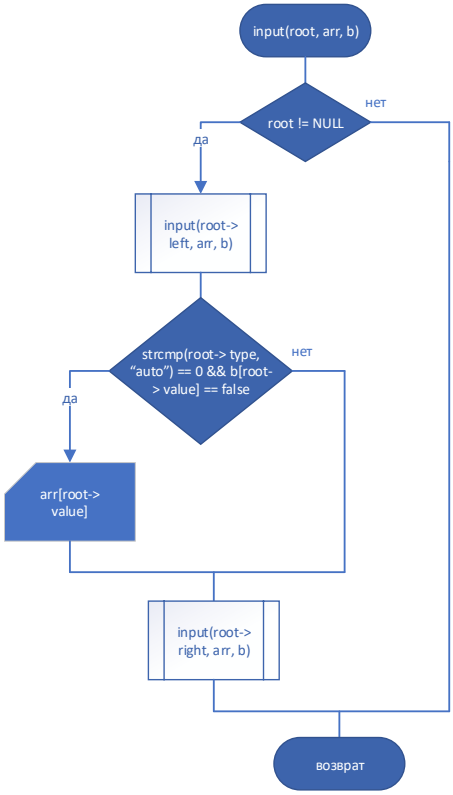

transform(str)
strchr("-+", *str) != NULL
нет
да
strcat(res, "0")
strchr("*/", *str) != NULL
нет
да
strcat(res, "1")
ptr = str + 1
ptr != '\0'
нет
да
strcat(res, ptr2)
return res
возврат
strchr("-+*", ptr) != 0 && (*(ptr - 1) == '(')
нет
да
stmcpy(str2, ptr2, (ptr – ptr2))
str2 + (ptr2 – ptr2) = '\0'
strcat(res, str2)
strchr("-+", *ptr) != 0
нет
да
strcat(res, "0")
strcat(res, "1")
ptr2 = ptr
strchr("-+*/)", *ptr) == 0 && strchr("-+*/(", *(ptr - 1) == 0)
нет
да
stmcpy(str2, ptr2, (ptr – ptr2))
str2 + (ptr – ptr2) = '\0'
strcat(res, str2)
strcat(res, "*")
ptr2 = ptr
ptr = ptr + 1

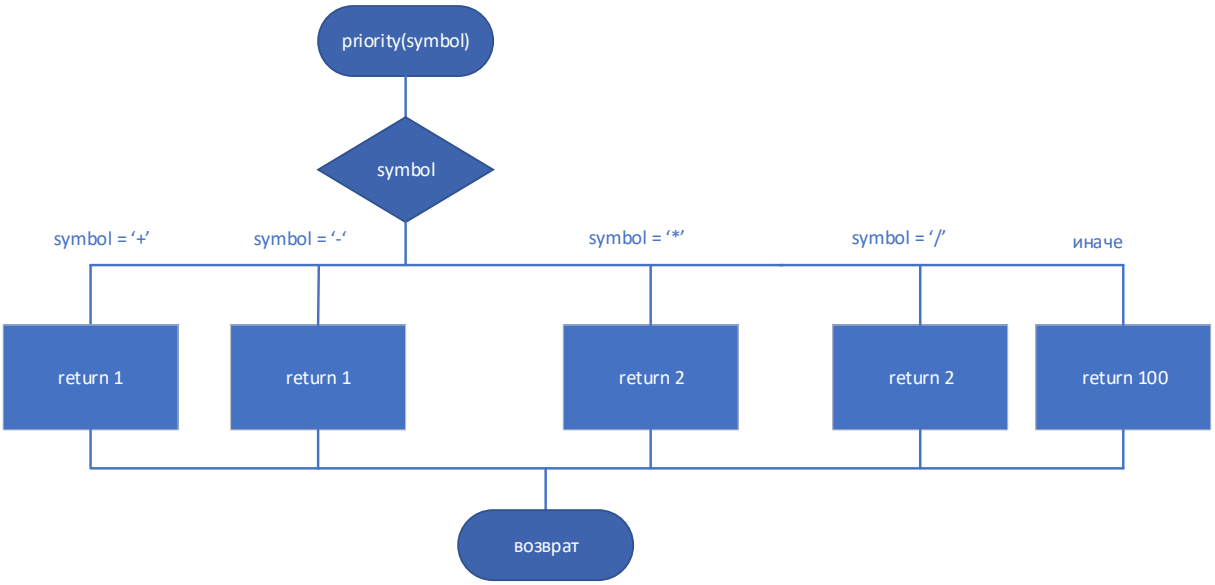

create_tree(str, first, last)
lincorrect_string (str, first, last)
нет
да
return create_tree(str, first + 1, last – 1)
count = 0
first == last
нет
да
strchr("0123456789", *(str + first)) != NULL
нет
да
strcpy(root-> type, "int")
strcpy(root-> type, "auto")
root->value = str + first
root->left = NULL
root->right = NULL
return root
str2 = str + first
i = first, last
str[i] == '('
нет
да
count = count + 1
str[i] == ')'
нет
да
count = count - 1
count == 0 && priority(str[i]) <= priority(str2)
нет
да
str2 = str + i
strcpy(root-> type, "operation")
root->value = str2
root->left = create_tree(str, first, ((str2 – str) – 1))
root->right = create_tree(str, ((str2 – str) + 1), last)
возврат

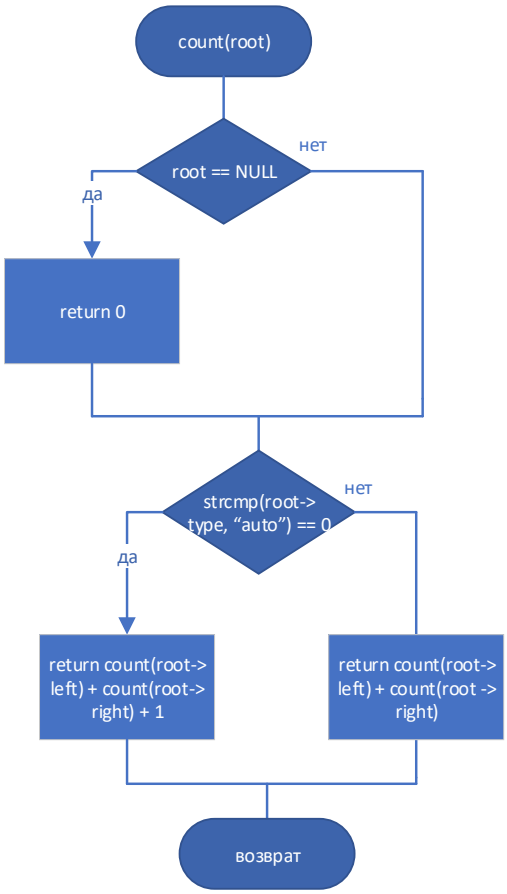

incorrect_string(str, first, last)
count = 0
ptr = str + first, str + last
return true
возврат
ptr == '('
нет
да
count = count + 1
count == 0
нет
да
return false
ptr == ')'
нет
да
count = count - 1

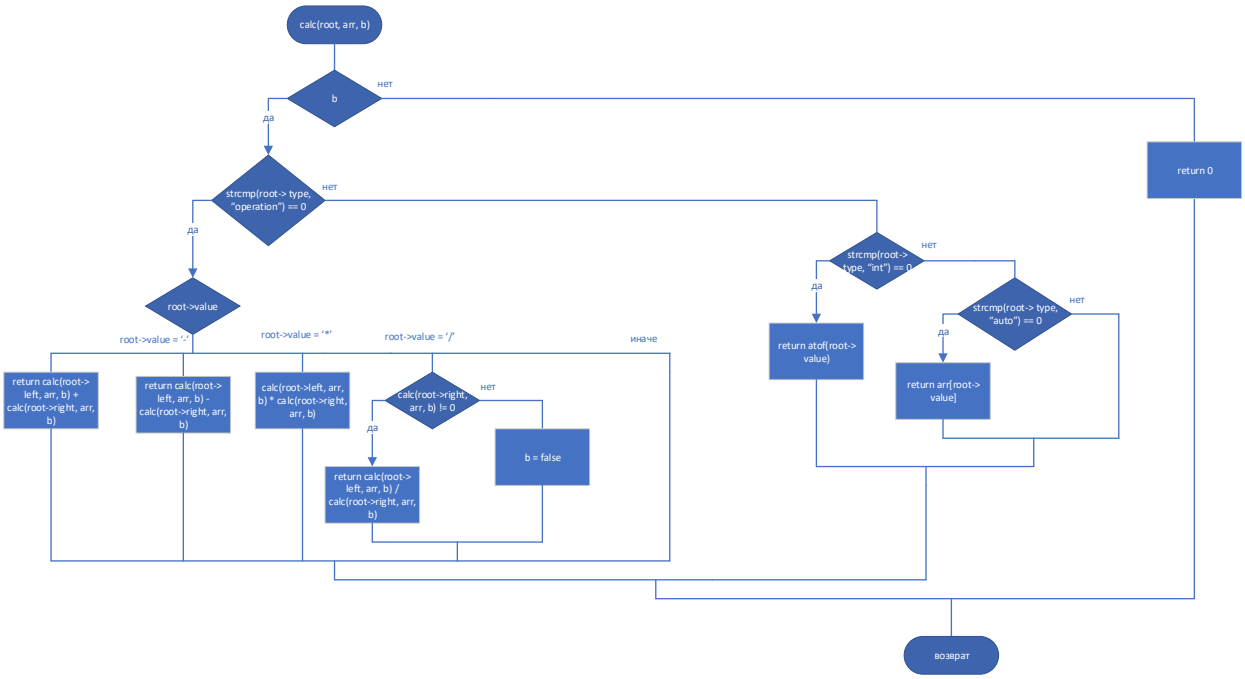

input(root, arr, b)
root != NULL
нет
да
input(root-> left, arr, b)
strcmp(root-> type, "auto") == 0 && b[root-> value] == false
нет
да
arr[root-> value]
input(root-> right, arr, b)
возврат

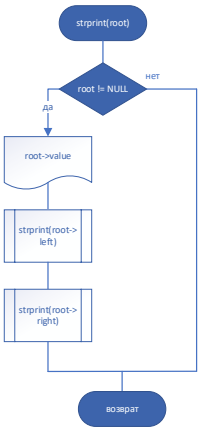

incorrect_string(str, first, last)
count = 0
ptr = str + first, str + last
return true
возврат
ptr == '('
нет
да
count = count + 1
count == 0
нет
да
return false
ptr == ')'
нет
да
count = count - 1

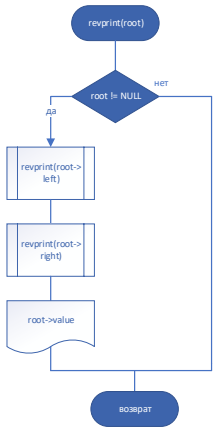

input(root, arr, b)
root != NULL
нет
да
input(root-> left, arr, b)
strcmp(root-> type, "auto") == 0 && b[root-> value] == false
нет
да
arr[root-> value]
input(root-> right, arr, b)
возврат

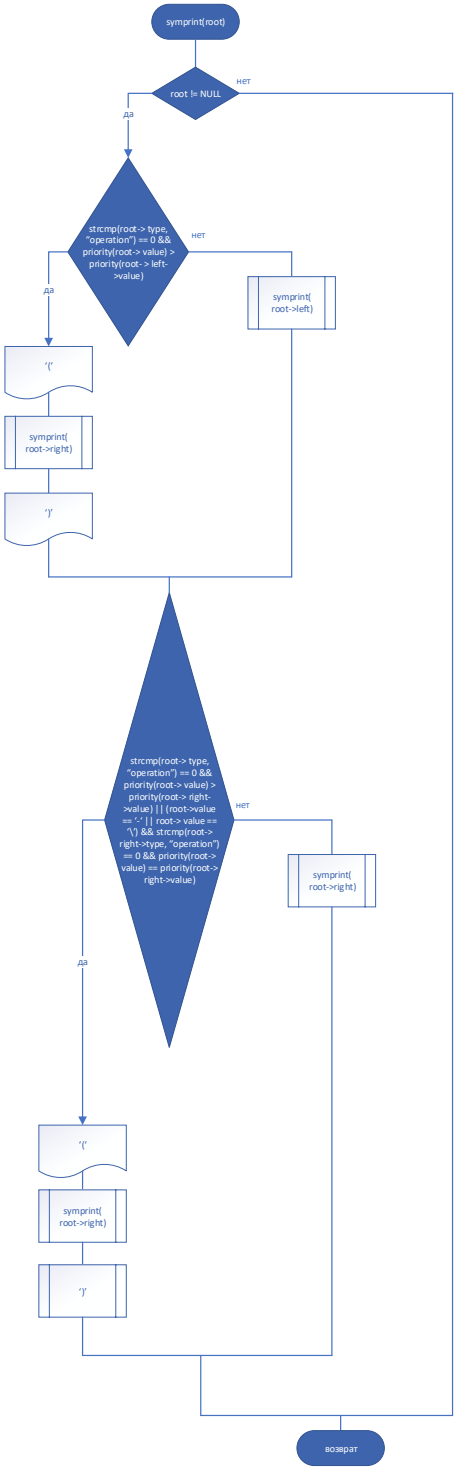

priority(symbol)
symbol
symbol = '+'
symbol = '-'
symbol = '*'
symbol = '/'
иначе
return 1
return 1
return 2
return 2
return 100
возврат


count(root)
root == NULL
нет
да
return 0
strcmp(root->type, "auto") == 0
нет
да
return count(root->left) + count(root->right) + 1
return count(root->left) + count(root -> right)
возврат


calc(root, arr, b)
b
нет
да
return 0
strcmp(root-> type, "operation") == 0
нет
да
root->value
root->value = '-'
root->value = '*'
root->value = '/'
иначе
return calc(root-> left, arr, b) + calc(root->right, arr, b)
return calc(root-> left, arr, b) - calc(root->right, arr, b)
calc(root->left, arr, b) * calc(root->right, arr, bj
calc(root->right, arr, b) != 0
нет
да
return calc(root-> left, arr, b) / calc(root->right, arr, b)
b = false
strcmp(root-> type, "int") == 0
нет
да
return atof(root-> value)
strcmp(root-> type, "auto") == 0
нет
да
return arr[root-> value]
возврат


strprint(root)
root != NULL
нет
да
root->value
strprint(root->left)
strprint(root->right)
возврат


revprint(root)
root != NULL
нет
да
revprint(root->left)
revprint(root->right)
root->value
возврат


symprint(root)
root != NULL
нет
да
strcmp(root-> type, "operation") == 0 && priority(root-> value) > priority(root- > left->value)
нет
да
symprint(root->left)
'('
symprint(root->right)
')'
strcmp(root-> type, "operation") == 0 && priority(root-> value) > priority(root-> right->value) || (root->value == '-' || root-> value == '\') && strcmp(root-> right->type, "operation") == 0 && priority(root-> value) == priority(root-> right->value)
нет
symprint(root->right)
да
'('
symprint(root->right)
')'
возврат

Текст программы

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>

const int length = 128;

struct Tree {
	char* value;
	char* type;
	Tree* left;
	Tree* right;
};

int priority(char symbol) {
	switch (symbol) {
		case '+':
		case '-':
			return 1;
		case '*':
		case '/':
			return 2;
		default: return 100;
	}
}

bool incorrect_string(char* str, int first, int last) {
	int count = 0;
	for (char* ptr = str + first; ptr != (str + last + 1); ++ptr) {
		if (*ptr == '(')
			++count;

		if (count == 0)
			return false;

		if (*ptr == ')')
			--count;
	}
	return true;
}

Tree* create_tree(char* str, int first, int last) {
	if (!incorrect_string(str, first, last)) {
		int count = 0;
		Tree* rootptr = new Tree;
		Tree* root = rootptr;
		if (first == last)
		{
			root->type = new char[length];
			if (strchr("0123456789", *(str + first)) != NULL)
				strcpy(root->type, "int");

			else
				strcpy(root->type, "auto");

			root->value = new char;
			*(root->value) = *(str + first);
			root->left = NULL;
			root->right = NULL;
			return root;
		}

		char* str2 = str + first;
		for (int i = first; i <= last; ++i) {
```

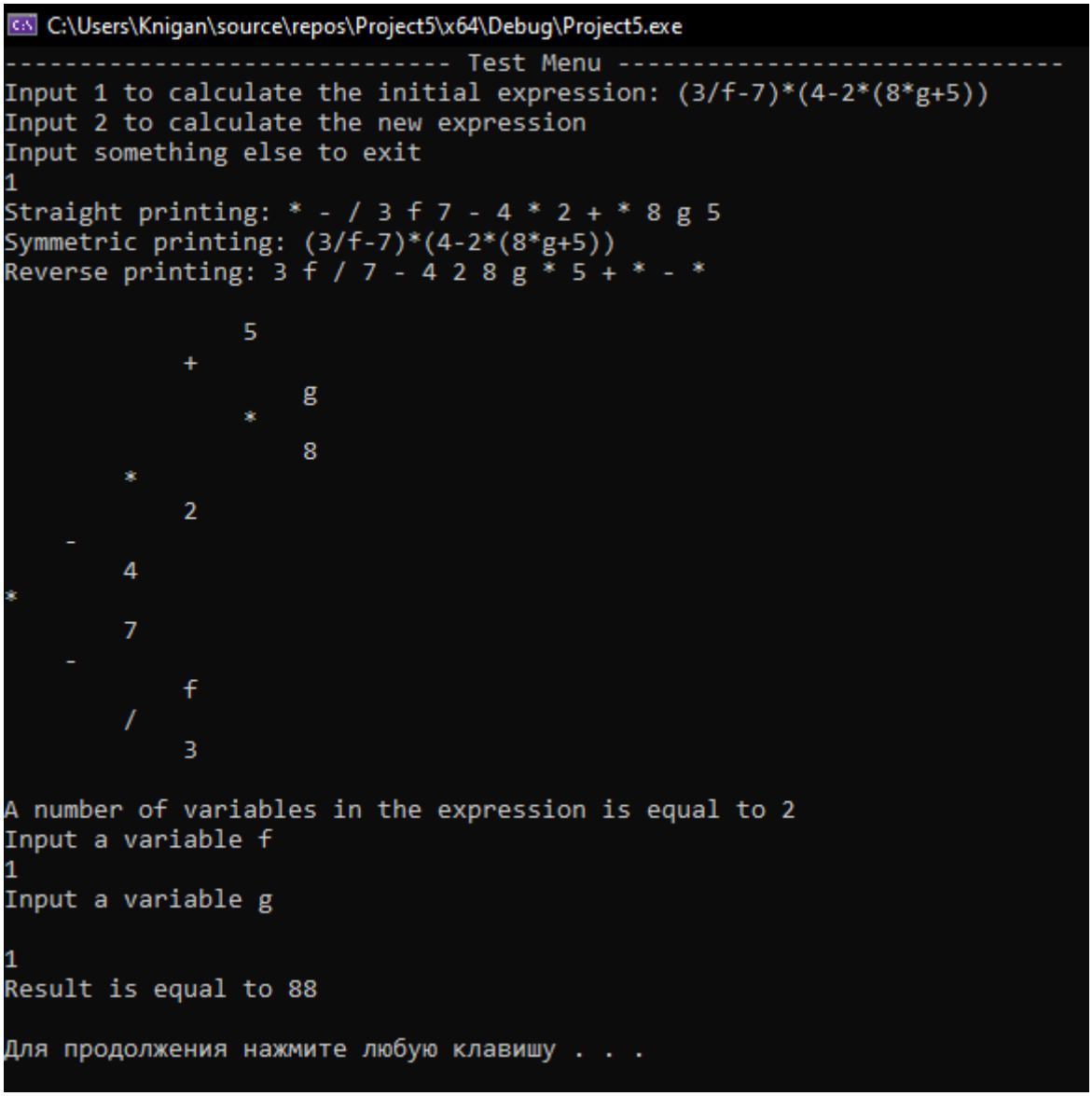
```
                        if (*(str + i) == '(')
                                count++;

                        else if (*(str + i) == ')')
                                count--;

                        if (count == 0 && priority(*(str + i)) <= priority(*str2))
                                str2 = str + i;
                }

                root->type = new char[length];
                strcpy(root->type, "operation");
                root->value = new char;
                *(root->value) = *str2;

                root->left = create_tree(str, first, ((str2 - str) - 1));
                root = rootptr;

                root->right = create_tree(str, ((str2 - str) + 1), last);
                root = rootptr;
        }
        else
                return create_tree(str, first + 1, last - 1);
}

void strprint(Tree* root) {
        if (root != NULL) {
                std::cout << *(root->value) << " ";
                strprint(root->left);
                strprint(root->right);
        }
}


void symprint(Tree* root)
{
        if (root != NULL)
        {
                if (strcmp(root->type, "operation") == 0 && priority(*(root->value)) >
priority(*(root->left->value))) {
                        std::cout << "(";
                        symprint(root->left);
                        std::cout << ")";
                }

                else
                        symprint(root->left);

                std::cout << *(root->value);

                if (strcmp(root->type, "operation") == 0 && ((priority(*(root->value)) >
priority(*(root->right->value))) || (((*(root->value) == '-')
                        || (*(root->value) == '/')) && ((strcmp(root->right->type,
"operation") == 0) && (priority(*(root->value)) == priority(*(root->right->value)))))))
                {
                        std::cout << "(";
                        symprint(root->right);
                        std::cout << ")";
                }
                else
                        symprint(root->right);
        }
}

void revprint(Tree* root) {
```

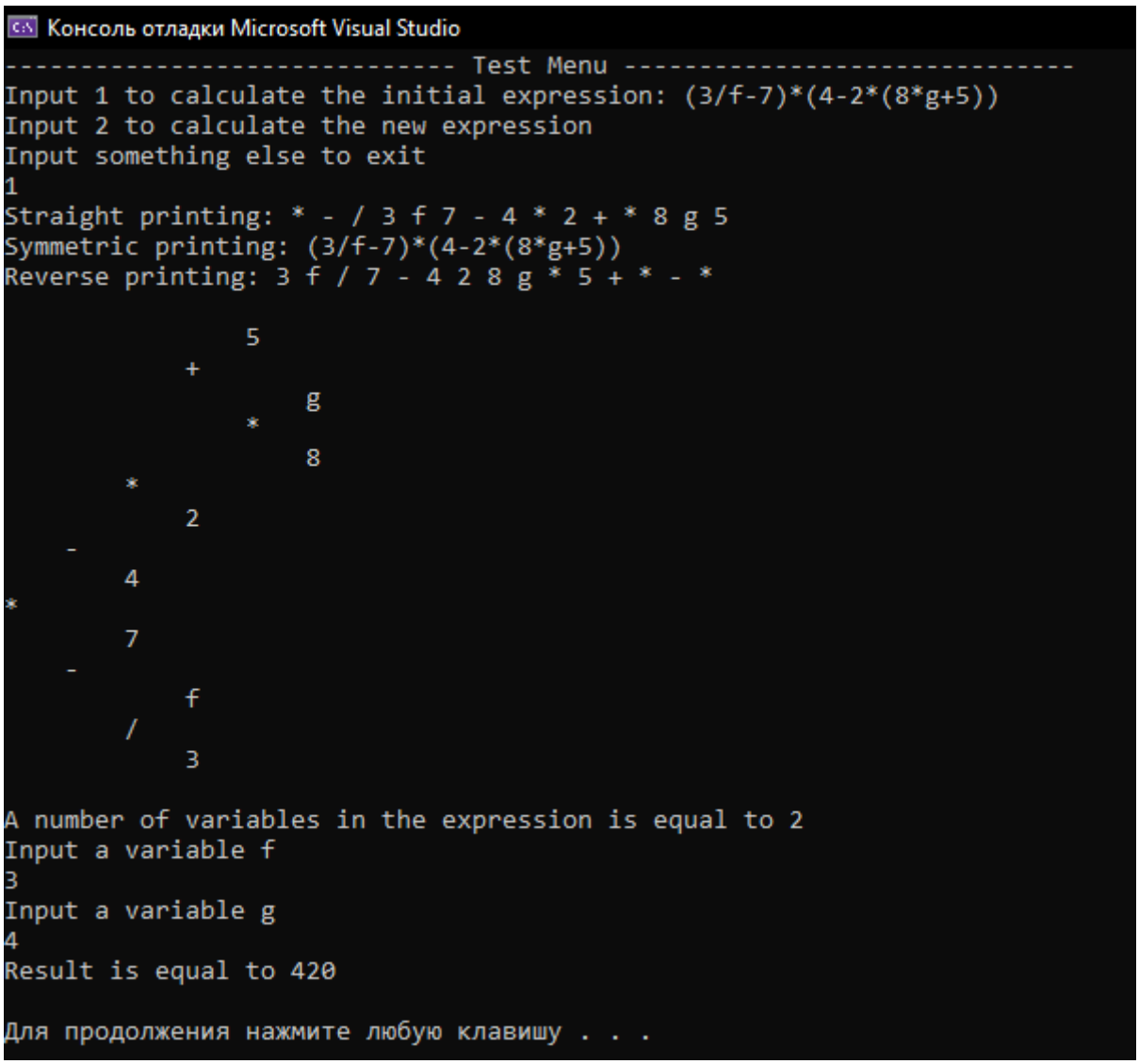
```
	if (root != NULL) {
		revprint(root->left);
		revprint(root->right);
		std::cout << *(root->value) << " ";
	}
}

void print_str(Tree* root) {
	if (root != NULL) {
		std::cout << "Straight printing: ";
		strprint(root);
		std::cout << std::endl;
		std::cout << "Symmetric printing: ";
		symprint(root);
		std::cout << std::endl;
		std::cout << "Reverse printing: ";
		revprint(root);
		std::cout << std::endl;
	}
}

void print_tree(Tree* root, int level) {
	if (root != NULL) {
		print_tree(root->right, level + 4);
		for (int i = 0; i < level; ++i) std::cout << " ";
		std::cout << *(root->value) << std::endl;
		print_tree(root->left, level + 4);
	}
}

void print(Tree* root) {
	print_str(root);
	std::cout << std::endl;
	print_tree(root, 0);
	std::cout << std::endl;
}

void input(Tree* root, double* arr, bool** b) {
	if (root != NULL) {
		input(root->left, arr, b);
		if (strcmp(root->type, "auto") == 0 && *(*b + *(root->value)) == false) {
			std::cout << "Input a variable " << *(root->value) << std::endl;
			std::cin >> *(arr + *(root->value));
			*(*b + *(root->value)) = true;
		}
		input(root->right, arr, b);
	}
}

double calc(Tree* root, double* arr, bool& b) {
	if (b) {
		if (strcmp(root->type, "operation") == 0) {
			switch (*(root->value)) {
				case '+': return calc(root->left, arr, b) + calc(root->right,
arr, b);
				case '-': return calc(root->left, arr, b) - calc(root->right,
arr, b);
				case '*': return calc(root->left, arr, b) * calc(root->right,
arr, b);
				case '/': {
					if (calc(root->right, arr, b) != 0)
						return calc(root->left, arr, b) / calc(root-
>right, arr, b);
					else b = false;
					break;
```

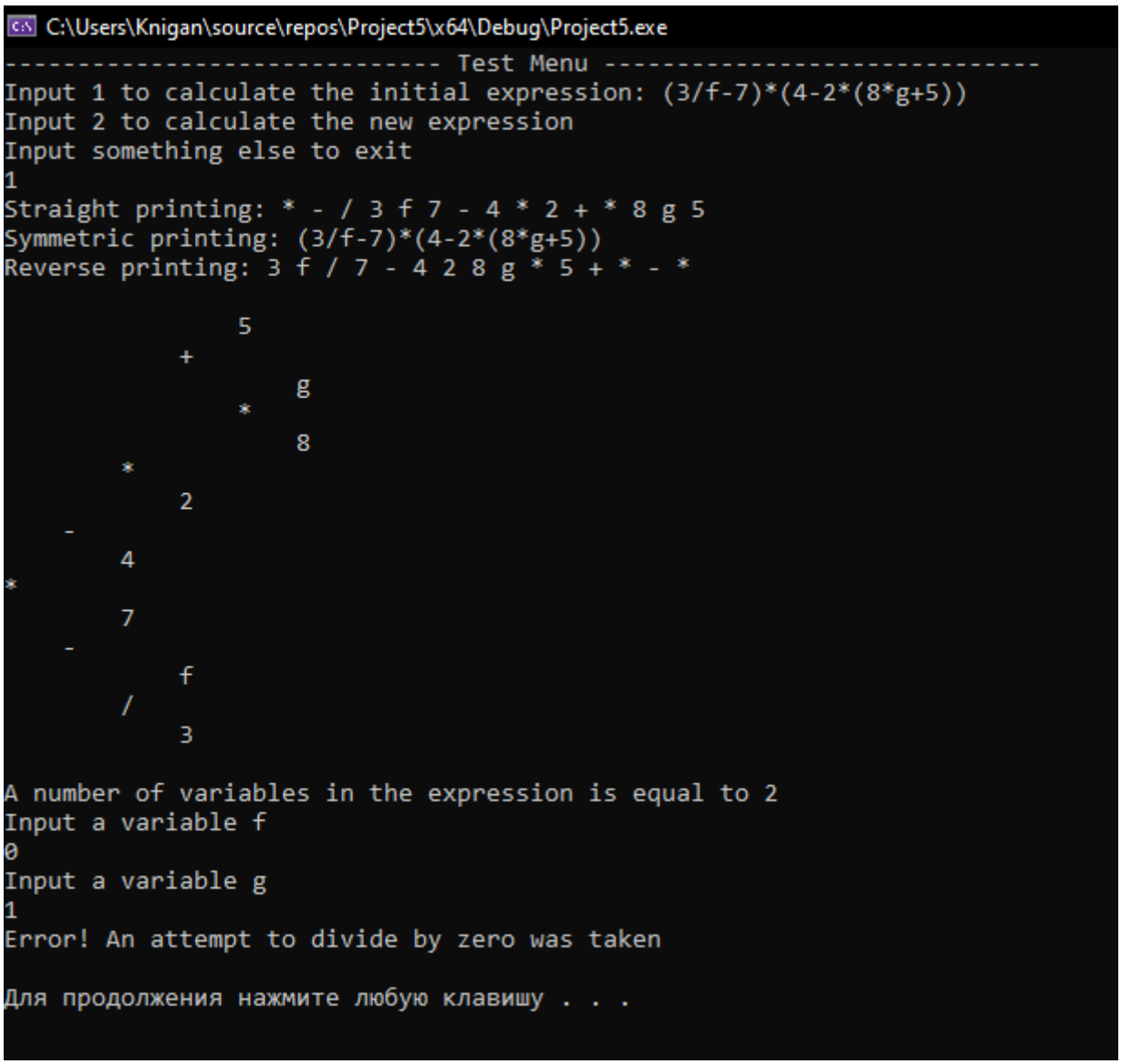
```
                        }
                        default: break;
                    }
                }
                else if (strcmp(root->type, "int") == 0)
                    return atof(root->value);

                else if (strcmp(root->type, "auto") == 0)
                    return *(arr + *(root->value));
        }
        else
                return 0;
}

char* transform(char* str) {
        char* res = new char[length];
        *res = '\0';

        if (strchr("-+", *str) != NULL)
                strcat(res, "0");

        if (strchr("*/", *str) != NULL)
                strcat(res, "1");

        char* str2 = new char[length];
        char* ptr2 = str;
        for (char* ptr = str + 1; *ptr != '\0'; ++ptr) {
                if (strchr("-+*", *ptr) != 0 && (*(ptr - 1) == '(')) {
                        strncpy(str2, ptr2, (ptr - ptr2));
                        *(str2 + (ptr - ptr2)) = '\0';
                        strcat(res, str2);
                        if (strchr("-+", *ptr) != 0)
                                strcat(res, "0");
                        else
                                strcat(res, "1");
                        ptr2 = ptr;
                }

                if (strchr("-+*/)", *ptr) == 0 && strchr("-+*/(", *(ptr - 1)) == 0) {
                        strncpy(str2, ptr2, (ptr - ptr2));
                        *(str2 + (ptr - ptr2)) = '\0';
                        strcat(res, str2);
                        strcat(res, "*");
                        ptr2 = ptr;
                }
        }
        strcat(res, ptr2);
        delete[] str2;
        return res;
}

int count(Tree* root) {
        if (root == NULL)
                return 0;

        if (strcmp(root->type, "auto") == 0)
                return count(root->left) + count(root->right) + 1;

        else
                return count(root->left) + count(root->right);
}

void deltree(Tree* root) {
        if (root != NULL) {
                deltree(root->left);
```

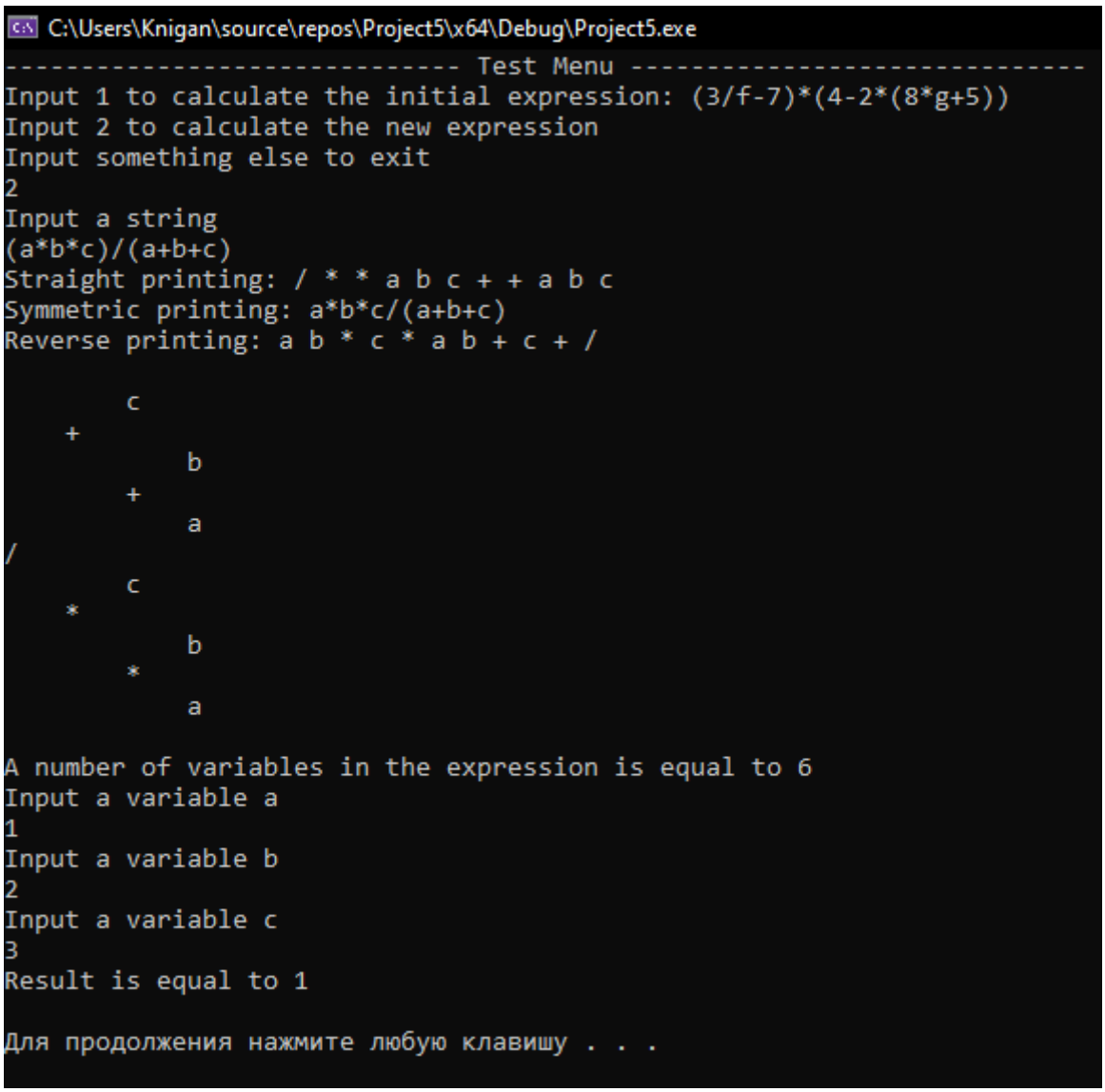
```
		deltree(root->right);
		delete root->value;
		delete root->type;
		delete root;
	}
}

void menu() {
	char* str = new char[length];
	*str = '\0';
	int key;
	std::cout << "----------------------------- Test Menu -------------------------
---\n";
	std::cout << "Input 1 to calculate the initial expression\n";
	std::cout << "Input 2 to calculate the new expression\n";
	std::cout << "Input something else to exit\n";
	std::cin >> key;
	switch (key) {
		case 1: strcpy(str, "((3/f-7)*(4-2*(8*g+5)))"); break;
		case 2: std::cout << "Input a string\n"; std::cin >> str; break;
		default: return;
	}

	Tree* root = NULL;
	str = transform(str);
	root = create_tree(str, 0, strlen(str) - 1);
	delete[] str;
	print(root);

	double* arr = NULL;
	bool** b = NULL;

	if (count(root) > 0) {
		std::cout << "A number of variables in the expression is equal to " <<
count(root) << std::endl;
		arr = new double[length];
		b = new bool*;
		*b = new bool[length];
		for (int i = 0; i < length; ++i)
			*(*b + i) = false;
		input(root, arr, b);
	}
	bool f = true;
	double res = calc(root, arr, f);
	if (f)
		std::cout << "Result is equal to " << res << std::endl;
	else
		std::cout << "Error! An attempt to divide by zero was taken\n";

	deltree(root);
	std::cout << std::endl;
}

void main() {
	menu();
	system("pause");
}
```

Тестирование



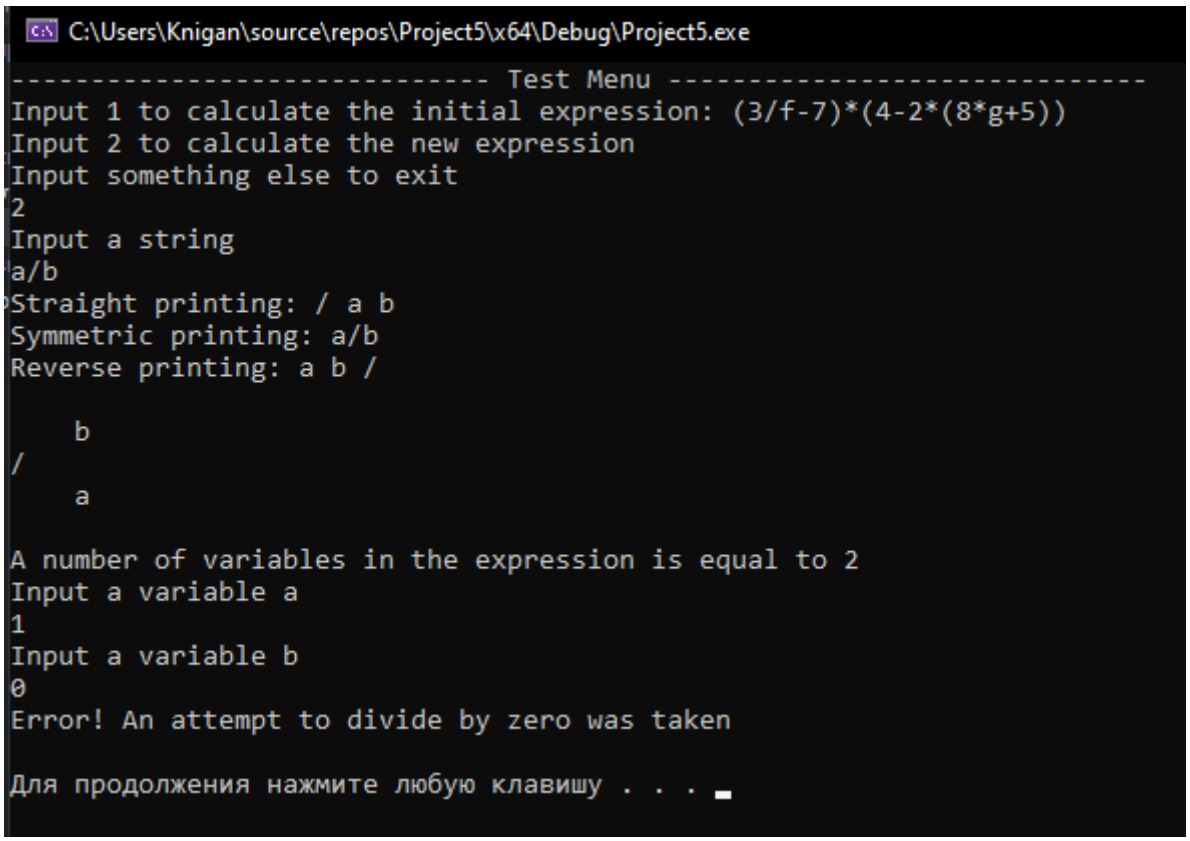

Консоль отладки Microsoft Visual Studio
------------------------------ Test Menu ------------------------------
Input 1 to calculate the initial expression: (3/f-7)*(4-2*(8*g+5))
Input 2 to calculate the new expression
Input something else to exit
1
Straight printing: * - / 3 f 7 - 4 * 2 + * 8 g 5
Symmetric printing: (3/f-7)*(4-2*(8*g+5))
Reverse printing: 3 f / 7 - 4 2 8 g * 5 + * - *
5
+
g
*
8
*
2
-
4
*
7
-
f
/
3
A number of variables in the expression is equal to 2
Input a variable f
3
Input a variable g
4
Result is equal to 420
Для продолжения нажмите любую клавишу . . .


C:\Users\Knigan\source\repos\Project5\x64\Debug\Project5.exe
------------------------------ Test Menu ------------------------------
Input 1 to calculate the initial expression: (3/f-7)*(4-2*(8*g+5))
Input 2 to calculate the new expression
Input something else to exit
1
Straight printing: * - / 3 f 7 - 4 * 2 + * 8 g 5
Symmetric printing: (3/f-7)*(4-2*(8*g+5))
Reverse printing: 3 f / 7 - 4 2 8 g * 5 + * - *
5
+
g
*
8
*
2
-
4
*
7
-
f
/
3
A number of variables in the expression is equal to 2
Input a variable f
0
Input a variable g
1
Error! An attempt to divide by zero was taken
Для продолжения нажмите любую клавишу . . .


C:\Users\Knigan\source\repos\Project5\x64\Debug\Project5.exe
---------- Test Menu ----------
Input 1 to calculate the initial expression: (3/f-7)*(4-2*(8*g+5))
Input 2 to calculate the new expression
Input something else to exit
2
Input a string
(a*b*c)/(a+b+c)
Straight printing: / * * a b c + + a b c
Symmetric printing: a*b*c/(a+b+c)
Reverse printing: a b * c * a b + c + /
c
+
b
+
a
/
c
*
b
*
a
A number of variables in the expression is equal to 6
Input a variable a
1
Input a variable b
2
Input a variable c
3
Result is equal to 1
Для продолжения нажмите любую клавишу . . .

```
C:\Users\Knigan\source\repos\Project5\x64\Debug\Project5.exe
------------------------------ Test Menu ------------------------------
Input 1 to calculate the initial expression: (3/f-7)*(4-2*(8*g+5))
Input 2 to calculate the new expression
Input something else to exit
2
Input a string
(a-b+c-d)/(-a+b-c+d)
Straight printing: / - + - a b c d + - + - 0 a b c d
Symmetric printing: (a-b+c-d)/(0-a+b-c+d)
Reverse printing: a b - c + d - 0 a - b + c - d + /

        d
    +
            c
        -
                b
            +
                    a
                -
                    0
/
        d
    -
            c
        +
                b
            -
                a

A number of variables in the expression is equal to 8
Input a variable a
5
Input a variable b
10
Input a variable c
15
Input a variable d
20
Result is equal to -1

Для продолжения нажмите любую клавишу . . .
```

```
C:\Users\Knigan\source\repos\Project5\x64\Debug\Project5.exe
------------------------------ Test Menu ------------------------------
Input 1 to calculate the initial expression: (3/f-7)*(4-2*(8*g+5))
Input 2 to calculate the new expression
Input something else to exit
2
Input a string
a*b*c/(a+b+c)
Straight printing: / * * a b c + + a b c
Symmetric printing: a*b*c/(a+b+c)
Reverse printing: a b * c * a b + c + /

        c
    +
            b
        +
            a
/
        c
    *
            b
        *
            a

A number of variables in the expression is equal to 6
Input a variable a
5
Input a variable b
6
Input a variable c
7
Result is equal to 11.6667

Для продолжения нажмите любую клавишу . . .
```



Вывод: я закрепил материал по динамическим структурам данных и научился работать с бинарными деревьями

Ответы на контрольные вопросы

1. Что такое синтаксическое дерево разбора?

Дерево синтаксического разбора – результат синтаксического анализа,

т.е. результат процесса сопоставления линейной последовательности лексем (слов, токенов) естественного или формального языка с его формальной грамматикой.

2. Какие основные способы разбора и расчета арифметических выражений?

Основной вариант разбора арифметического выражения – это представление его в виде двоичного дерева. Также существует разбор с помощью стэка.

3. Для каких случаев используется префиксная, инфиксная и постфиксная запись?

В зависимости от варианта обхода бинарного дерева можно получить разные формы записи выражения.

Префиксная запись – прямое прохождение дерева(корень – левое поддерево – правое поддерево).

Инфиксная запись – симметричный обход (левое поддерево – корень – правое поддерево). В трансляторах широко используется постфиксная запись выражений, которая получается в результате обратного обхода (левое поддерево - правое поддерево - корень)